eCOMMONS

Loyola University Chicago
Loyola eCommons

Computer Science: Faculty Publications and Other Works

Faculty Publications

2008

Combining SOA and BPM Technologies for Cross-System Process Automation

Sebastian Herr

John Shafaee

Konstantin Läufer
Loyola University Chicago, klaeufer@gmail.com

George K. Thiruvathukal
Loyola University Chicago, gkt@cs.luc.edu

Guido Wirtz

Recommended Citation

Herr, S., K. Läufer, J. P. Shafaee, G. K. Thiruvathukal, and G. Wirtz (2008). Combining SOA and BPM technologies for Cross-System process automation. In SEKE, pp. 339-344.

This Conference Proceeding is brought to you for free and open access by the Faculty Publications at Loyola eCommons. It has been accepted for inclusion in Computer Science: Faculty Publications and Other Works by an authorized administrator of Loyola eCommons. For more information, please contact ecommons@luc.edu.

This work is licensed under a Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 License.
Copyright © 2008 Sebastian Herr, Konstantin Läufer, John P. Shafaee, George K. Thiruvathukal, and Guido Wirtz

PROCEEDINGS

SEKE 2008

The 20th International Conference on Software Engineering & Knowledge Engineering

Sponsored by

Knowledge Systems Institute Graduate School, USA

Technical Program

July 1-3, 2008
Hotel Sofitel, Redwood City, San Francisco Bay, USA

Organized by

Knowledge Systems Institute Graduate School

Copyright © 2008 by Knowledge Systems Institute Graduate School

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the publisher.

ISBN 1-891706-22-5 (paper)

Additional Copies can be ordered from:
Knowledge Systems Institute Graduate School
3420 Main Street
Skokie, IL 60076, USA
Tel:+1-847-679-3135
Fax:+1-847-679-3166
Email:office@ksi.edu
http://www.ksi.edu

Proceedings preparation, editing and printing are sponsored by
Knowledge Systems Institute Graduate School

Printed by Knowledge Systems Institute Graduate School

Foreword

On behalf of the Program Committee Co-Chairs, who are listed below, and the Program Committee of the 2008 International Conference on Software Engineering and Knowledge Engineering (SEKE-2008), it is an honor to welcome you to SEKE-2008 in San Francisco, California. It has been my pleasure as Program Committee Chair to help organize this year's impressive scientific and technical program and the technical proceedings. The proceedings contain the papers selected for presentation at SEKE-2008. I hope these proceedings will serve as a valuable reference for the research community.

The International Conference on Software Engineering and Knowledge Engineering has entered its 20th year. For the past nineteen years, the Conference on Software Engineering and Knowledge Engineering has provided a unique, centralized, forum for academic and industrial researchers and practitioners to discuss the application of either software engineering methods in knowledge engineering or knowledge-based techniques in software engineering. Preference is given to papers that emphasize the transference of methods between both engineering disciplines; however, outstanding papers on software engineering or knowledge engineering alone have also been presented.

This year's program committee consists of the following great team of Vice Program chairs:

Program Co-Chairs:

Guido Wirtz, Bamberg University, Germany
Jerry Gao, San Jose State University, USA
Du Zhang, California State University, USA

The SEKE-2008 Program Committee selected papers for publication in the proceedings and presentation at the Conference based upon a rigorous review process of the full papers. We received an overwhelming 252 submissions from many countries. The acceptance rate for full papers is 48% and for short papers is 16%. This year, authors from 37 countries (Australia, Austria, Brazil, Canada, China, Colombia, Czech Republic, Denmark, Egypt, Finland, France, Germany, India, Iran, Ireland, Italy, Japan, Jordan, Malta, Mexico, Netherlands, Pakistan, Portugal, Singapore, South Korea, Spain, Sri Lanka, Sweden, Switzerland, Taiwan, Thailand, Tunisia, Turkey, United Kingdom, United States, Uruguay, Venezuela) will present papers at the conference.

I appreciate having had the opportunity to serve as the Program Chair for this Conference, and am very grateful for the outstanding efforts provided by the Program Committee Co-Chairs. The Program Committee members and reviewers provided excellent support in promptly reviewing the manuscripts. I want to extend my sincere and deepest thanks to Dr. Shihong Huang and Dr. Masoud Sadjadi as the Publicity Co-Chairs, Dr. Jose' Carlos Maldonado as the South America Liaison. My appreciation also goes to the keynote speakers for sharing their insights and experiences with the conference attendees, and to the SECIML 2008 workshop and special tracks organizers. I am grateful to the authors and sessions chairs for their time and efforts to make SEKE-2008 a success. As always, Dr. S. K. Chang of the Knowledge Systems Institute, USA, provided excellent guidance throughout the effort. Last but not the least, we all owe a special debt of gratitude the heroic efforts of Mr. Daniel Li, of the Knowledge Systems Institute.

Finally, I truly hope that you will enjoy the technical programs of SEKE-2008 and encourage you to explore and enjoy the various attractions San Francisco has to offer.

Taghi M. Khoshgoftaar
SEKE-2008 Program Chair

The 20th International Conference on Software Engineering & Knowledge Engineering (SEKE 2008)

July 1-3, 2008
Hotel Sofitel, Redwood City, San Francisco Bay, USA

Conference Organization

Steering Committee Chair

Shi-Kuo Chang, *University of Pittsburgh, USA*

Steering Committee

Vic Basili, *University of Maryland, USA*
Bruce Buchanan, *University of Pittsburgh, USA*
C. V. Ramamoorthy, *University of California, Berkeley, USA*

Conference Chair

Daniel Cooke, *Texas Tech University, USA*

Program Chair

Taghi M. Khoshgoftaar, *Florida Atlantic University, USA*

Program Co-Chairs

Jerry Gao, *San Jose State University, USA*
Guido Wirtz, *Bamberg University, Germany*
Du Zhang, *California State University, USA*

Program Committee

Alain Abran, *Universite du Quebec,Canada*
Silvia Teresita Acuna, *Universidad Autnoma De Madrid, Spain*
Edward B. Allen, *Mississippi State University, USA*
Mikhail Auguston, *Naval Postgraduate School, USA*
Doo-Hwan Bae, *Computer Science Dept. KAIST, Korea*
Xiaoying Bai, *Tsinghua University, China*
Maria Teresa Baldassarre, *University of Bari, Italy*
Luciano Barezi, *Politechnico de Milano, Italy*
Emese Bari, *eBay Inc., USA*
Saida Benlarbi, *Alcatel-Lucent, Canada*
Sami Beydeda, *The Federal Finance Office, Germany*
Alessandro Bianchi, *University of Bari, Italy*
Jim Bieman, *Colorado State University, USA*
Gary D. Boetticher, *University of Huston Clear Lake, USA*
Jean-Michel Bruel, *University of Pau, France*
Barrett Bryant, *University of Alabama , Birmingham, USA*
Kai-Yuan Cai, *Beijing University of Aeronautics and Astronautics, China*
Danilo Caivano, *University of Bari, Italy*
Gerardo Canfora, *University of Sannio, Italy*
Joao W. Cangussu, *University of Texas at Dallas, USA*
Giovanni Cantone, *University of Rome Tor Vergata, Italy*
Jeffrey C. Carver, *Mississippi State University, USA*
Christine W. Chan, *University of Regina, Canada*
Keith C.C. Chan, *The Hong Kong Polytechnic University, Hong Kong*
W.K. Chan, *City University of Hong Kong, Hong Kong*
Kuang-Nan Chang, *Eastern Kentucky University, USA*
Ned Chapin, *InfoSci Inc., USA*
Shu-Ching Chen, *Florida International University, USA*
Yinong Chen, *Arizona State University, USA*
Harry Cheng, *University of California, Davis, USA*
Yoonsik Cheon, *University of Texas at El Paso, USA*
Peter J. Clarke, *Florida International University, USA*
Panos Constantopoulos, *Athens University of Economics, Greece*
Kendra Cooper, *University of Texas at Dallas, USA*
Maria Francesca Costabile, *University of Bari, Italy*
Juan J. Cuadrado-Gallego, *University of Alcala, Spain*
Alfredo Cuzzocrea, *ICAR Institute and DEIS Department, University of Calabria, Italy*
Scott Dick, *University of Alberta, Canada*
Jin Song Dong, *National University of Singapore, Singapore*
Jing Dong, *University of Texas at Dallas, USA*
Philippe Dugerdil, *HEG-University of Applied Sciences, Switzerland*
Reiner Dumke, University of Magdeburh, Germany
Schahram Dustdar, *University of Technology Vienna, Austria*
Christof Ebert, *Vector Consulting Services GmbH, Germany*
Faezeh Ensan, *University of New Brunswick, Canada*
Behrouz Homayoun Far, *University of Calgary, Canada*
Martin S. Feather, *Jet Propulsion Laboratory, USA*
Robert Feldt, *Blekinge Institute of Technology, Sweden*
Norman Fenton, *Queen Mary University of London, UK*

Eduardo B. Fernandez, *Florida Atlantic University, USA*
Todd Fitch, *Intuit Corp, USA*
Andres Folleco, *Florida Atlantic University, USA*
Jose Fortes, *University of Florida, USA*
Renata Fortes, *University of Sao Paulo, Brazil*
Kehan Gao, *Eastern Connecticut State University, USA*
Alessandro Garcia, *Lancaster University, UK*
Felix Garcia, *University of Castilla-La Mancha, Spain*
Carlo Ghezzi, *Politechnico di Milano Technical University, Italy*
Holger Giese, *University of Potsdam, Germany*
Itana Gimenes, *UEM/PR/Brazil, Brazil*
Swapna Gokhale, *University of Connecticut, USA*
Jeff Gray, *University of Alabama , Birmingham, USA*
Des Greer, *Queens University Belfast, UK*
Eric Gregoire, *Universite dArtois, France*
Paul Grunbacher, *Johannes Kepler University Linz, Austria*
Mark Harman, *Kings College London, UK*
Ahmed E. Hassan, *Queens University, Canada*
Xudong He, *Florida International University, USA*
Rattikorn Hewett, *Texas Tech University, USA*
Mei Hsing, *Fu Jen Catholic University, Taiwan*
Shihong Huang, *Florida Atlantic University, USA*
Byung-Yeon Hwang, *The Catholic University of Korea, Korea*
Ali Idri, *ENSIAS, Rabat, Morocco*
Peter In, *Korea University, Korea*
Natalia Juristo, *Madrid Technological University, Spain*
Gunes Koru, *University of Maryland, Balt Cty, USA*
Mark Last, *Ben-Gurion University of the Negev, Israel*
Jeff Lei, *University of Texas at Arlington, USA*
Tao Li, *Florida International University, USA*
Yingdar Lin, *National Chiao-Tung University, Taiwan*
Xiaodong Liu, *Napier University, UK*
Yan (Jenny) Liu, *National ICT, Australia*
Yi Liu, *Georgia college and State University, USA*
Jian Lu, *Nanjing University, China*
Zhongyu (Joan) Lu, *The University of Huddersfield, UK*
Heiko Ludwig, *IBM TJ Watson Research Center, Almaden, San Jose, USA*
Michael R. Lyu, *Chinese University of Hong Kong, Hong Kong*
Jose Carlos Maldonado, University of Sao Paulo, Brazil
Antonio Mana, *University of Malaga, Spain*
Emilia Mendes, *University of Auckland, New Zealand*
Harald Meyer, *HPI Potsdam, Germany*
Rym Mili, *University of Texas at Dallas, USA*
James Miller, *University of Alberta, Canada*
Henry Muccini, *Univerita degli Studi de L Aquila, Italy*
Nachi Nagappan, *Microsoft Research, Seattle, USA*
Martin Neil, *MQueen Mary (U. of London), UK*
Allen Nikora, *Jet Propulsion Laboratory, USA*
Elisabetta Di Nitto, *Politechnico de Milano, Italy*
Mehmet Orgun, *Macquarie University, Australia*
Manish Parashar, *Rutgers University, USA*

Witold Pedrycz, *University of Alberta, Canada*
Jun Peng, *Chongqing University of Science and technology, China*
Massimiliano Di Penta, *University of Sannio, Italy*
Hoang Pham, *Rutgers University, USA*
Rajeev Raje, *Indiana University-Purdue University, Indianapolis, USA*
Sanjay Ranka, *University of Florida, USA*
Marek Reformat, *University of Alberta, Canada*
Robert Reynolds, *Wayne State University, USA*
Daniel Rodriguez, *The University of Alcala, Spain*
George Roussos, *University of London, UK*
Guenther Ruhe, *University of Calgary, Canada*
Masoud Sadjadi, *Florida International University, USA*
Ramon Sagarna, *The University of Birmingham, UK*
Ahmed Salem, *California State University at Sacramento, USA*
Naeem Seliya, *University of Michigan at Dearborn, USA*
Tony Shan,*Wachovia Bank, USA*
Yidong Shen, *Chinese Academy of Science, China*
Martin Shepperd, *Brunel University, UK*
Simon Shim, *SAP Lab. LLC, USA*
Michael Shin, *Texas Tech University, USA*
George Spanoudakis, *City University, UK*
Arndt von Staa, *PUC-Rio, Brazil*
Mark Stamp, *San Jose State University, USA*
Nenad Stankovic, *Univ. of Aizu, Japan*
Xiao Su, *San Jose State University, USA*
Rajesh Subramanyan, *Siemens Corporate Research, Inc. USA*
Jeff Tian, *Southern Methodist University, USA*
Genny Tortora, *University of Salerno, Italy*
Peter Troger, *Blekinge Institute of Technology, Sweden*
T.H. Tse, *University of Hong Kong, Hong Kong*
Bhekisipho Twala, *Pretoria, South Africa*
Michael VanHilst, *Florida Atlantic University, USA*
Silvia Regina Vergilio, *UFPR, Brazil*
Marlon Vieira, *Siemens Corporate Research, Inc. USA*
Qianxiang Wang, *Beijing University, China*
Yingxu Wang, *University of Calgary, Canada*
Christiane Gresse von Wangenheim, *Universidate do Vale do Itaja, Brazil*
Tim Weitzel, *Bamberg University, Germany*
Laurie Williams, *North Carolina State University, USA*
Victor Winter, *University of Nebraska at Omaha, USA*
Eric Wong, *University of Texas at Dallas, USA*
Ye Wu, *Advanced Information Technology Center, SAIC, USA*
Baowen Xu, *Southeast University, China*
Zhiwei Xu, *University of Michigan at Dearborn, USA*
Hongji Yang, *De Montfort University, UK*
Huiqun Yu, *East China University of Science and Technology, China*
Cui Zhang, *California State University, USA*
Zhi-Hua Zhou, *Nanjing University, China*
Hong Zhu, *Oxford Brookes University, UK*
Xingquan Zhu, *Florida Atlantic University, USA*

Eugenio Zimeo, *University of Sannio, Italy*
Andrea Zisman, *City University, UK*

Publicity Co-Chairs

Shihong Huang, *Florida Atlantic University, USA*
Masoud Sadjadi, *Florida International University, USA*

South America Liasion

Jose Carlos Maldonado, *University of Sao Paulo, Brazil*

Industry Advisory Committee

Silvia Ahmed, *NetApp, USA*
Emese Bari, *eBay, USA*
Cecilia Claudio, *SVP/CIO Information Technology, SanDisk Corporation, USA*
Yi Deng, *Dean, School of Computer Science, Florida International University, USA*
Todd Fitch, *Intuit Corp, USA*
Chuck Fredrick, *Chief Technology Officer, Douglas County, Denver, USA*
Laura Haas, *Distinguished Engineer and Director, Computer Science, Almaden Research Center, IBM*
J. S. Ke, *Senior Fellow, Institute for Information Industry, Taiwan*
Shilpa Kolhatkar, *Cisco, USA*
Gerry Pompa, *Vice President, Compunetix, USA*
A. J. Rhem, *Senior Partner, A. J. Rhem and Associates Inc., USA*

Proceedings Cover Design

Gabriel Smith, *Knowledge Systems Institute Graduate School, USA*

Conference Secretariat

Judy Pan, *Chair, Knowledge Systems Institute Graduate School, USA*
Jerilyn Tinio, *Knowledge Systems Institute Graduate School, USA*
Jessica Braunstein, *Knowledge Systems Institute Graduate School, USA*
Chen-Cheang Huang, *Knowledge Systems Institute Graduate School, USA*
Daniel Li, *Knowledge Systems Institute Graduate School, USA*

Table of Contents

Foreword iii

Conference Organization iv

How to make an information elephant dance

Dr. Cecilia Claudio 1

Impact! The Challenge of Industrial Research in Computer Science in a web 2.0 world

Dr. Laura Haas 2

Building Global Ecosystem for Collaborative Computing Research and Education

Dr. Yi Deng 3

Applications I

Transformations for Rapid Prototyping of Time-critical Applications
Shi-Kuo Chang, Zhoulan Zhang, Colin J. Ihrig, Paolo Maresca, Valentina Ternelli … 4

Case Study: Applying Business Process Management Systems (S)
Gregor Scheithauer, Guido Wirtz 12

Verification of Optimization Algorithms: a Case Study of a Quadratic Assignment Problem Solver
Tsong Yueh Chen, Huimin Lin, Robert Merkel, Daoming Wang 16

Software Engineering Methodology I

Towards a Theoretical Model for Evaluating the Acceptance of Model-driven Measurement Procedures (S)
Nelly Condori-Fernandez, Oscar Pastor …… 22

Knowledge Transformation from Task Scenarios to View-based Design Diagrams
Nima Dezhkam, Kamran Sartipi …….. 26

PSPCAT: A PSP Data Collection and Analysis Tool (S)
Chien-Hung Liu, Shu-Ling Chen, Yu-Chun Huang …… 33

Software Process Modeling I

A Systematic Method for Process Tailoring Based on Knowledge Reuse (S)
Xiao-yang He, Ya-sha Wang, Yu-xin Teng, Jin-gang Guo …….... 38

Linking Return on Training Investment with Defects Causal Analysis
Santiago Matalonga, Tomas San Feliu Gilabert …….... 42

Autonomous Reconfiguration Procedures for EJB-based Enterprise Applications
Thomas Vogel, Jens Bruhn, Guido Wirtz ……... 48

Software Maintenance and Evolution

Cross-language Clone Detection
Nicholas A. Kraft, Brandon W. Bonds, Randy K. Smith …….... 54

Software Maintenance Maturity Model (S3^{m}DSS) A Decision Support System
Alain April, Naji Habra, Arnaud Counet …….... 60

Odyssey-MEC: Model Evolution Control in the Context of Model-Driven Architecture
Chessman Correa, Leonardo Murta, Claudia Werner ……... 67

SE with Computational Intelligence and Machine Learning I

Analyzing the Impact of Attribute Noise on Software Quality Classification
Andres A. Folleco, Taghi M. Khoshgoftaar, Lofton A. Bullard …… 73

An Adaptive Neural Network with Dynamic Structure for Software Defect Prediction
Zhiwei Xu, Naeem Seliya, Weibiao Wu …….. 79

Software Engineering Methodology II

Evaluating the Accuracy of Call Graphs Extracted with the Eclipse CDT
Nicholas A. Kraft, Kevin S. Webb …….... 85

A Comparison of Time Tracking Tools for Software Developers
Jouni Lappalainen, Lasse Harjumaa, Jukka Sirvio, Tytti Pokka, Heidi Moisanen, Hanna Leskinen …….. 91

RealSpec: an Executable Specification Language for Modeling Resources
Amir A. Khwaja, Joseph E. Urban …….. 97

Software Testing I

Predicting Change Propagation in Object-oriented Systems: a Control-call Path Based Approach and Associated Tool
Linda Badri, Mourad Badri, Daniel St-Yves …….. 103

A Qualitative Assessment of the Reverse Engineering Capabilities of Unit Testing Tools for Understanding Java Programs
Andy Tinkham, Scott Tilley, Tauhida Parveen …….. 111

Estimating Event Lifetimes for Distributed Runtime Verification
Christos Kloukinas, George Spanoudakis, Khaled Mahbub …….. 117

SE with Computational Intelligence and Machine Learning II

Ontology-learning Supported Sematic Search Using Cooperative Agents
Cheng Zhong, Zilan (Nancy) Yang, Mohsen Afsharchi, Behrouz H. Far ……... 123

Automating a Domain Model Aware Reengineering Methodology
Javier Belmonte, Philippe Dugerdil …… 129

Explaining Product Release Planning Results Using Concept Analysis
Gengshen Du, Thomas Zimmermann, Guenther Ruhe …….. 137

Weighted Static Code Attributes for Software Defect Prediction
Burak Turhan, Ayse Bener …….. 143

Software Engineering Methodology III

Predicting Software Project Size Using Project Generated Information
Marcio de O. Barros …… 149

Supporting Reusable Component Selection with Use Case Gap-based Development Effort Estimation
Xin Zhou, Bonnie Ray, Chenhua Feng ……... 155

A Project Scheduling Method Based on Human Resource Availability
Lizi Xie, Junchao Xiao, Dapeng Liu, Qing Wang …….. 161

Estimating the Effort of Independent Verification and Validation in the Context of Mission-critical Software Systems - A Case Study (S)
Haruka Nakaoa, Adam Trendowicz, Jurgen Munch ……..... 167

Software Process Modeling II

Unified Basic Concepts for Process Capability Models
Clenio F. Salviano, Adriana M. C. M. Figueiredo …… 173

Systematic Approach to Risk Management in Software Projects through Process Tailoring
Lisandra M. Fontoura, Roberto Tom Price ……... 179

Process tailoring based on well-formedness rules
Eliana B. Pereira, Ricardo M. Bastos, Toacy C. Oliveira …….. 185

Non-invasive Software Process Data Collection for Expert Identification
Andrea Janes, Alberto Sillitti, Giancarlo Succi ……. 191

SOA-Based Software Testing and Maintenance

Using XML Patterns to Guide Perturbation Based Testing of Web Services
Paulo N. Cruz Filho, Silvia Regina Vergilio ……... 197

Translating OWL Specified Domain Knowledge to Aspect Oriented Model
Juanzi Li, Xinyu You, Xiaoying Bai ……..... 203

MAPLE: a Maintenance Approach for Pattern-enabLed rEconfiguration of SOA-based Enterprise Application
Songlin Hu, Ying Liang, Jiuming Tian, Yicheng Song …… 209

Reliability Oriented QoS Driven Composite Service Selection Based on Performance Prediction (S)
Lei Yang, Yu Dai, Bin Zhang ……. 215

Service Oriented Technology and Web Technology I

Design of an RSS Crawler with Adaptive Revisit Manager (S)
Bum-Suk Lee, Jin Woo Im, Byung-Yeon Hwang, Du Zhang ……..... 219

QuickPay Online Payment Protocol (S)
Jian Dai, Mark Stamp ……. 223

Sharing Application Logic Across Programming Language Boundaries (S)
Dennis S. Patrone, Bina Ramamurthy ……..... 227

Software Reuse and Component Technology I

Synergizing Collaboration and Reuse in Software Engineering (S)
Stefan Seedorf, Oliver Hummel ……... 232

Improving Component Container Development Process through Product Line Engineering
Guoliang Liu, Yang Li, Jun Wei ……. 238

System and Software Architecture I

.NET Extensions to the π-architecture Description Languages *(S)*
Zawar Qayyum, Flavio Oquendo …… 244

Towards Collaborative Development Based on Software Architecture (S)
Yanchun Sun, Hui Song, Xinghua Wang, Wenpin Jiao ……... 250

Choosing a Software Architecture: An Approach and a Case Study
C. Ghezzi, G. Tamburrelli ……... 255

Formal Methods I

PROTEF: Automatic Verification of Pattern-Based LTL Templates
Luis Garcia, Steve Roach, Salamah Salamah …… 261

Formal Specification of Object-oriented Systems with Collaborative Objects and Petri Nets – a Case Study
Boleslaw Mikolajczak ……... 267

A Property Specification Tool for Generating Formal Specifications: Prospec 2.0
Irbis Gallegos, Omar Ochoa, Ann Gates, Steve Roach, Salamah Salamah, Corina Vela 273

SE with Computational Intelligence and Machine Learning III

On the Rarity of Fault-prone Modules in Knowledge-based Software Quality Modeling
Taghi M. Khoshgoftaar, Naeem Seliya, Dennis J. Drown …… 279

Machine Learning and Value-based Software Engineering: a Research Agenda
Du Zhang ……. 285

Automatic Clustering of Defect Reports
Vasile Rus, Sameer Mohammed, Sajjan Shiva ……... 291

Software Engineering Methodology IV

Subjective Assessment of the Mutual Influence of ISO 9126 Software Qualities: an Empirical Study
Sandro Morasca 297

Reverse Engineering Interface Protocols for Comprehension of Large C++ Libraries during Code Evolution Tasks
Edward B. Duffy, Jason O. Hallstrom, Brian A. Malloy 303

Knowledge Management to Support the Deployment of a CMMI Level 3 Process
A. P. Cavalcanti, F. Furtado, V. Moura, R. Costa, S. R. L. Meira 309

System and Software Architecture II

Code Transformation Techniques and Management Architecture for Self-manageable Distributed Applications
M. Muztaba Fuad 315

A Decision-centric Architecture Design Method Facilitating the Contextually Capture and Reuse of Design Knowledge
Xiaofeng Cui, Yanchun Sun, Sai Xiao, Hong Mei 321

System Architecture Induces Document Architecture (S)
Peter Henderson, Nishadi De Silva 327

A Software Framework for Integrative Physiological Model Simulation (S)
E. Zeynep Erson, M. Cenk Cavusoglu 333

Service Oriented Technology and Web Technology II

Combining SOA and BPM Technologies for Cross-System Process Automation
S. Herr, K. Laufer, J. Shafaee, G. K. Thiruvathukal, G. Wirtz 339

Ontology-Enabled Generation of Embedded Web Services
Klaus Marius Hansen, Weishan Zhang, Goncalo Soares 345

Modeling Services to Construct Service-oriented Healthcare Architecture for Digital Home-care Business
Chi-Lu Yang, Yeim-Kuan Chang, Chih-Ping Chu …… 351

Databases

Testing Relational Database Schemas with Alternative Instance Analysis
Maria Claudia F. P. Emer, Silvia Regina Vergilio, Mario Jino ……... 357

Analyzing Termination and Confluence in Active Rule Base via a Petri Net Approach (S)
Lorena Chavarria-Baez, Xiaoou Li …… 363

A Fuzzy Trigger Language for Relational Database Systems
Ying Jin, Tejaswitha Bhavsar …….. 367

Data Mining I

A Comparative Study on Data Representation to Categorize Text Documents (S)
D.A. Meedeniya, A.S. Perera
…….... 371

An Example on Economics-driven Software Mining
Rami Bahsoon, Wolfgang Emmerich …….. 375

VP: an Efficient Algorithm for Frequent Itemset Mining
Qin Ding, Wen Shen Huang …….... 381

Model-Based Software Engineering I

Evolution Shelf: Exploiting Evolution Styles within Software Architectures
Olivier Le Goaer, Mourad-Chabane Oussalah, Dalila Tamzalit, Abdelhak-Djamel Seriai …… 387

Coverage-based Testing Using Qualitative Reasoning Models
Harald Brandl, Gordon Fraser, Franz Wotawa …….. 393

Traceability Models to Control an Aspectual Model-driven Development (S)
Marta S. Tabares, Raquel Anaya, Ana Moreira, Joao Araujo, Fernando Arango …….. 399

Knowledge Engineering

Knowledge-based System Development with Scripting Technology: A Recommender System Example
Dietmar Jannach …….. 405

Integrating Trust Management into Usage Control in P2P Multimedia Delivery
Li Yang, Raimund Ege …… 411

Flow Balancing Model for Air Traffc Flow Management (S)
Bueno Borges de Souza, Li Weigang, Antonio Marcio Ferreira Crespo, Victor Rafael Rezende Celestino …… 417

Applications II

VisRFID: Visualizing Customer Behavior in Geotemporal Space Using RFID Technology
Beomjin Kim, Keith Bock, Michael Burton, Rod Strong, Benjamin Aeschliman …… 422

Analyzing Manufacturing Process Knowledge Flows with KoFI
Oscar M. Rodriguez-Elias, Alberto L. Moran, Jaqueline I. Lavandera, Aurora Vizcaino …… 428

Performance: a Longitudinal Study
Nenad Stankovic …….. 434

Formal Methods II

A Formal Approach for Translating a SAM Architecture to PROMELA
Gonzalo Argote-Garcia, Peter J. Clarke, Xudong He, Yujian Fu, Leyuan Shi …… 440

An Algorithm for Computing Loop Functions
Ali Mili, Shir Aharon, Chaitanya Nadkarni …….... 448

Verifying Behavioral Correctness of Design Pattern Implementation
Tu Peng, Jing Dong, Yajing Zhao …….. 454

Software Requirements Engineering I

Automated Multiperspective Requirements Traceability Using Ontology Matching Technique
Namfon Assawamekin, Thanwadee Sunetnanta, Charnyote Pluempitiwiriyawej ……... 460

Eliciting Scenarios from Scenarios
Abdolmajid Mousavi, Behrouz H. Far ……... 466

Tailoring an Aspectual Goal-oriented Approach to Model Features
Carla Silva, Fernanda Alencar, Joao Araujo, Ana Moreira, Jaelson Castro …….. 472

Representing Textual Requirements as Graphical Natural Language for UML Diagram Generation
Magda G. Ilieva, Harold Boley ……... 478

Software Testing II

A Dynamic Adjusting Method for Test Case Prioritization (S)
Bo Qu, Changhai Nie, Baowen Xu, Xiaofang Zhang …… 484

A Systematic Mapping Study on Non-Functional Search-based Software Testing
Wasif Afzal, Richard Torkar, Robert Feldt …… 488

A Degraded ILP Approach for Test Suite Reduction
Zhenyu Chen, Xiaofang Zhang, Baowen Xu …….. 494

A Meta-model to Support Regression Testing of Web Applications
Yanelis Hernandez, Tariq M. King, Jairo Pava, Peter J. Clarke …….. 500

Service Oriented Technology and Web Technology III

Service Granularity Effects in SOA
Ned Chapin ……..…… 506

Securing Service-oriented Systems Using State-Based XML Firewall
Abhinay Reddyreddy, Haiping Xu ……... 512

Toward Model Checking Web Services Over the Web
John C. Sloan, Taghi M. Khoshgoftaar …….. 519

A Metadata Model for Managing and Querying XML Resources in Peer-to-peer Systems
Deise de Brum Saccol, Nina Edelweiss, Renata de Matos Galante ……... 525

Formal Methods III

Minimal Observability for Transactional Hierarchical Services
Debmalya Biswas, Blaise Genest …… 531

Using Boolean Cardinality Constraint for LTS Bounded Model Checking (S)
Sachoun Park, Gihwon Kwon …….. 537

Japanese Puzzle as a SAT Problem (S)
Sachoun Park, Gihwon Kwon …….. 543

Business Models for Service-Oriented Architectures

Bridging the Semantic Gap Between Process Documentation and Process Execution
Gregor Scheithauer, Guido Wirtz, Candemir Toklu …….. 549

Performance Challenges in Migrating to SOA Based Healthcare Systems
Suyog Gaidhani, Vijayananda Jagannatha ……... 555

Agent-Based Technology and Intelligence I

Dynamically Optimize Process Execution Based on Process-agent (S)
Jian Dai, Junchao Xiao, Qing Wang, Mingshu Li, Huaizhang Li 561

Mobile-FIRST: a Mobile Agent Based First Responder System (S)
Jason Honda, Harry H. Cheng, Donna Djordjevich 565

Ontology-based and Evolutionary Search for Computational Agents Schemes (S)
Roman Neruda 569

Software Reuse and Component Technology II

A Goal-oriented Mixed-granularity Component Selection Method for Huge Component Repositories (S)
Xiaolin Xi, Jiyong Park, Jiakun Liu, Seongsoo Hong 573

A Case Study: Self-managed COTS Component-based Elevator System (S)
Michael E. Shin, Fernando Paniagua 577

Using Scenario Monitoring to Address State Based Crosscutting Concerns (S)
Mark Mahoney, Tzilla Elrad 581

Methods and Tools for Robust Services and Service Compositions

Negotiating Service Levels - A Generic Negotiation Framework for WS Agreement
Sebastian Hudert, Heiko Ludwig, Guido Wirtz 587

Taxonomy on Consistency Requirements in the Business Process Integration Context
Andreas Schönberger, Guido Wirtz 593

Developing Enterprise Applications with Support to Dynamic Unanticipated Evolution (S)
Hyggo O. de Almeida, Marcos F. Pereira, Marcio de M. Ribeiro, Angelo Perkusich, Emerson Loureiro, Evandro Costa 599

Data Mining II

Privacy-preserving Classification of Data Streams (S)
Ching-Ming Chao 603

Comparing the Use of Traditional and Associative Classifiers towards Personalized Recommendations
Joel Pinho Lucas, Saddys Segrera, María N. Moreno 607

Discovering Meaningful Clusters from Mining the Software Engineering Literature
Yan Wu, Harvey Siy, Li Fan 613

Model-Based Software Engineering II

A Model-Driven Approach for the Semi-automated Generation of Web-based Applications from Requirements
Ali Fatolahi, Stephane S. Some, Timothy C. Lethbridge 619

A Model-driven Toolset to Support an Approach for Analyzing Integration of Business Process Aspect of Enterprise Application Integration
Souvik Barat, Vinay Kulkarni 625

Model-based Test Complexity Analysis for Software Installation Testing (S)
Jerry Gao, Karen Kwok, Todd Fitch 631

Service Oriented Technology and Web Technology IV

A Similarity Analysis Model for Semantic Web Information Filtering Applications
Lucas Drumond, Rosario Girardi, Fabio Silva 638

Fuzziness in the Semantic Web: Survey and Future Directions
Seyed Koosha Golmohammadi, Marek Reformat, Witold Pedrycz 643

A Language-based Approach to Addressing Reliability in Composite Web Services
Onyeka Ezenwoye, S. Masoud Sadjadi 649

Agents, Web, and Security

A Systematic Process for Domain Engineering
Eduardo Santana de Almeida, Alexandre Alvaro, Vinicius Cardoso Garcia, Daniel Lucredio, Renata Pontin de Mattos Fortes, Silvio Romero de Lemos Meira ……... 655

Diagnosing Runtime Violations of Security & Dependability Properties
Theocharis Tsigritis, George Spanoudakis …… 661

Model-Based Software Engineering III

Translating Workflow Diagrams into Web Designs
Antonio Navarro, Jorge Merino, Alfredo Fernandez-Valmayor, Jesus Cristobal ……... 667

A Security Domain Model for Static Analysis and Verification of Software Programs
Alan B. Shaffer …… 673

Component Based Architectures for eXtreme Transacion Processing (S)
Luca Vetti Tagliati …….... 679

Ontologies I

An Ontology for Controlled Experiments on Software Engineering
Rogerio Eduardo Garcia, Erika Nina Hohn, Ellen Francine Barbosa, Jose Carlos Maldonado ……..…….. 685

Improving Automatic Model Creation Using Ontologies
Sven J. Korner, Tom Gelhausen …….. 691

Ontology-based Development of Testing Related Tools
Ellen F. Barbosa, Elisa Y. Nakagawa, Ana C. Riekstin, Jose C. Maldonado ……... 697

Software Test Automation and Practice I

Test Order Generation for Efficient Object-oriented Class Integration Testing
Rattikorn Hewett, Phongphun Kijsanayothin, Darunee Smavatkul …….. 703

Using Observer Automata to Select Test Cases for Test Purposes
Gordon Fraser, Martin Weiglhofer, Franz Wotawa ……... 709

Building Testable Components - a Systematic Approach and Its Experimental Study
Jerry Gao, Wrihang Roberto Liang, Radhika Chhabra, Ramyashree Swamyo, Ma Xiang ……..……... 715

SyncTest: a Tool to Synchronize Source Code, Model and Testing
Xiaoying Bai, Tao Liu …….. 723

Agent-Based Technology and Intelligence II

A Virtual Machine for Distributed Agent-oriented Programming
Bin Zhou, Hong Zhu …….... 729

MAAEM: a Multi-agent Application Engineering Methodology
Adriana Leite, Rosario Girardi, Uiratan Cavalcante …… 735

A Semantic Based Certification and Access Control Approach Using Security Patterns on SEAGENT
Fatih Tekbacak, Tugkan Tuglular, Oguz Dikenelli …….... 741

Documenting and Modeling Multi-agent Systems Product Lines
Ingrid Nunes, Uira Kulesza, Camila Nunes, Carlos J. P. de Lucena …….... 745

Model-Based Software Engineering IV

A Study of the Model Explosion Problem in CTL Model Update
Yulin Ding, Yan Zhang …….... 752

Feature Modeling for Context-Aware Software Product Lines
Paula Fernandes, Claudia Werner, Leonardo Murta ……. 758

MEtaGile: A Pragmatic Domain-specific Modeling Environment (S)
Olivier Buchwalder, Claude Petitpierre ……. 764

Software Requirements Engineering II

Obtaining Well-Founded Practices about Elicitation Techniques by Means of an Update of a Previous Systematic Review (S)
Oscar Dieste, Marta Lopez, Felicidad Ramos …… 769

Automatic Discovery of Interactions Between Software Requirements
Edgar S. Calisaya, Marcos R. S. Borges, Maria Luiza M. Campos …….. 773

A Model-driven Approach for Software Product Lines Requirements Engineering
Mauricio Alferez, Uira Kulesza, Andre Sousa, Joao Santos, Ana Moreira, Joao Araujo, Vasco Amaral …….. 779

Model Interpretation for Executable Observation Specifications (S)
Mathias Funk, Piet van der Putten, Henk Corporaal …… 785

Security Technology & Systems

Network Intrusion Detection Based on Bayesian Networks (S)
Alma Cemerlic, Li Yang, Joseph M. Kizza …….. 791

Supremum of Agent Number Needed in Analyzing Security Protocols Based on Horn Logic
Feng Liu, Zhoujun Li, Ti Zhou, Mengjun Li …… 795

Towards the Detection of Emulated Environments via Analysis of the Stochastic Nature of System Calls
Tauhida Parveen, William Allen, Scott Tilley, Gerald Marin, Richard Ford …….... 802

SE of Autonomic Grid Computing Systems and Applications I

Self-managed Deployment in a Distributed Environment via Utility Functions
Debzani Deb, Michael J. Oudshoorn, John Paxton …….. 808

Design of a Fault-tolerant Job-flow Manager for Grid Environments Using Standard Technologies, Job-flow Patterns, and a Transparent Proxy
Gargi Dasgupta, Onyeka Ezenwoye, Liana Fong, Selim Kalayci, S. Masoud Sadjadi, Balaji Viswanathan …….. 814

Supporting Context-awareness in Web-based Groupware Development (S)
Jose Maria N. David, Marcos R. S. Borges, Jose A. Pino …….. 820

Software Engineering Methodology V

Object-Z to Java/OO-Perl: A Conversion from Object-Z to Executable Skeletal Code with Dynamically Checkable Design Contracts
Sherri M. Sanders, Cui Zhang …… 824

An Empirical Study on Modularization of Object Oriented Software
Jing Liu, Bin Liu, Chi K. Tse, Keqing He …….. 830

Bridging the Gap Between Slicing and Model-based Diagnosis
Franz Wotawa ……..…… 836

Dynamic Analysis and Design Pattern Detection in Java Programs (S)
Lei Hu, Kamran Sartipi ……..…….. 842

Service Oriented Technology and Web Technology V

Active Ontologies - an Approach for Using Ontologies as Semantic Web Services Interfaces (S)
Tiago Cordeiro Marques, Marcio Gurjao Mesquita, Julio Cesar Campos Neto, Pedro Porfirio Muniz Farias …….. 847

Failure Prediction Based Self-healing Approach for Web Service Composition (S)
Yu Dai, Lei Yang, Bin Zhang, Kening Gao …… 853

A Wed-based data Management and Analysis System for CO2 Capture (S)
Yuxiang Wu, Christine W. Chan …… 857

Software Test Automation and Practice II

Integrating Random Testing with Constraints for Improved Efficiency and Diversity
Yoonsik Cheon, Antonio Cortes, Gary T. Leavens, Martine Ceberio ……... 861

Properties of Machine Learning Applications for Use in Metamorphic Testing
Christian Murphy, Gail Kaiser, Lifeng Hu, Leon Wu ……... 867

Fault Injection Testing of User-space File Systems Using Traditional and Aspect-based Techniques (S)
Jonathan Hittle, Sudipto Ghosh …….. 873

Evaluation of Personalized Information Systems: Application in Intelligent Transport System (S)
M. Soui, C. Kolski, M. Abed, G. Uster …… 877

SE of Autonomic Grid Computing Systems and Applications II

Dynamis: Dynamic Overlay Service Composition for Distributed Stream Processing
Farshad A. Samimi, Philip K. McKinley …… 881

Wings4Symbian: A Pervasive Computing Middleware for Symbian OS Mobile Devices
Olympio C. Silva Filho, Danilo F. S. Santos, Angelo Perkusich, Emerson Loureiro, Hyggo Almeida ……..……... 887

An OWL/SWRL Based Diagnosis Approach in a Pervasive Middleware
Weishan Zhang, Klaus Marius Hansen …….. 893

Model-Based Software Engineering V

A Constraint Model for Automated Deployment of Automotive Control Software
Mihai Nica, Bernhard Peischl, Franz Wotawa ……... 899

Applying Critical Pair Analysis in Graph Transformation Systems to Detect Syntactic Aspect Interaction in UML State Diagrams
Zaid Altahat, Tzilla Elrad, Luay Tahat 905

Model Comparison: a Strategy-Based Approach
Kleinner Oliveira, Toacy Oliveira 912

Ontologies II

Towards Metrics for Ontology Balance
Steffen Mencke, Martin Kunz, Reiner R. Dumke 918

Techniques for De-fragmenting Mobile Applications: A Taxonomy
Damith C. Rajapakse 923

Identifying NFRs Conflicts Using Quality Ontology
Taiseera Al Balushi, Pedro R. Falcone Sampaio, Mitul Patel, Oscar Corcho, Pericles Loucopoulos 929

Ontology-based Process Modeling and Execution Using STEP/EXPRESS
Arndt Muhlenfeld, Wolfgang Mayer, Franz Maier, Markus Stumptner 935

Reviewer's Index 941

Author's Index 944

Note: (S) means short paper.

Keynote I:
Why Doesn't the Software Do What I Need It to Do?
or Aligning IT Objectives with the Business

Cecilia Claudio

Abstract

Software Development has a long and varied history that is littered with many failures and fewer successes. While Software Developers have the best intentions the "results" when viewed from a management perspective often fall short. Schedule and budget overruns are common and often the original business objectives (and certainly the expectations) are not or only partially met. This presentation will examine the historical reasons that have resulted in unmet expectations and various strategies will be discussed for ensuring that expectations are aligned with results.

About Dr. Cecilia Claudio

Cecilia Claudio has spent the last 15 years as CIO at leading organizations and has established a proven track record of transforming IT to become a valued partner to business users. She is currently CIO at SanDisk Corporation, which she joined in February 2007. Previously Ms. Claudio was CIO of Mercury Interactive; Zurich Financial Services; Anthem Blue Cross/Blue Shield and Harvard Pilgrim Health Care.

Ms. Claudio also serves on several boards including Sybase, newScale and DiVitas and has previously served on the board of Farmers Group Inc. She has also served on CIO Advisory boards for Cisco, IBM, Siebel, Syndera and Tablus. She was recognized by Computerworld as one of the Premier 100 IT Executives worldwide in 2001, and has been recognized as one of the Top 100 Women in Computing in the US by McGraw Hill.

Ms. Claudio earned an MA in Philosophy in 1974 in Lisbon - Portugal.

Keynote II:
Impact! The Challenge of Industrial Research in Computer Science in a Web 2.0 World

Laura Haas

Abstract

The IBM Almaden Research Center, located in San Jose, California, is one of eight IBM Research Division facilities worldwide and a premier industrial research laboratory. Currently, over 200 permanent members, postdoctoral scholars, and academic visitors pursue research in computer science and closely related fields. But pressures on industrial research in computer science are increasing. Labs need to be nimble to adapt and to continue to bring value to their company and its customers. In this talk, these pressures and the resulting demands on industrial labs are illustrated through an exploration of several research projects from Almaden's computer science department. We discuss how they are changing IBM's business, their disciplines, and maybe even the world, and the diverse mechanisms they employ to have impact. Projects will be drawn from five different areas in which Almaden specializes, including computer science theory, information management, health informatics, human-computer interaction and services science, and will illustrate a number of different approaches from traditional, academic-style research to "standard" industrial research directly influencing IBM's products to research in the marketplace (featuring work directly with customers, partners and standards bodies) to spin-outs to engagement with educators to influence curriculum formation.

About Dr. Laura Haas

Laura Haas is an IBM Distinguished Engineer and Director of Computer Science at Almaden Research Center. She also leads information management research worldwide across IBM’s eight research labs. Most recently, she was responsible for Information Integration Solutions (IIS) architecture in IBM's Software Group, after leading the IIS development team through its first two years. Dr. Haas joined the development team in 2001 as manager of DB2 UDB Query Compiler development. Previously, Dr. Haas was a research staff member and manager at IBM's Almaden Research Center for nearly twenty years. In Research, she worked on and managed a number of exploratory projects in distributed database systems. She is best known for her work on the Starburst query processor (from which DB2 UDB was developed), on Garlic, a system which allowed federation of heterogeneous data sources, and on Clio, the first semi-automatic tool for heterogeneous schema mapping. Garlic technology married with DB2 UDB query processing is the basis for WebSphere Information Server's federation capabilities, while Clio capabilities are a core differentiator for the new Rational Data Architect. Dr. Haas is an active member of the database community, serving as vice chair of ACM SIGMOD from 1989-1997, and, currently, as Vice President of the VLDB Board of Trustees, as well as on many program committees for technical conferences. She has received several IBM awards for Outstanding Technical Achievement, and an IBM Corporate Award for her work on federated database technology. She is a member of the IBM Academy of Technology, an ACM Fellow, and a member of the board of the Computing Research Association.

Keynote III:
Building Global Ecosystem for Collaborative Computing Research and Education

Yi Deng

Abstract

I will present our experiences in developing international partnerships for collaborative, interdisciplinary computing research and education, from the Latin American Grid Consortium (LA Grid) to the NSF sponsored International Partnership for Research and Education (PIRE) initiative, involving leading universities, industry and governmental organizations in eight countries on four continents. I will discuss our partnership model, our problem-driven research framework, as well as our approach of using research collaboration to drive streamlined education and workforce development.

About Dr. Yi Deng

Yi Deng received his Ph.D. in Computer Science from the University of Pittsburgh in 1992. He currently serves as the Dean of School of Computing and Information Sciences at the Florida International University (FIU) – the state university of Florida in Miami, a position he has held since 2002. He is an accomplished leader in computing research and innovation, and has led many large scale multidisciplinary research and education initiatives. He founded and directed three research centers, including the Center for Advanced Distributed System Engineering, the NSF Center of Emerging Technologies for Advanced Information Processing and High Confidence Systems, and the IBM Center for Autonomic and Grid Computing at FIU. He co-founded the Latin American Grid (LA Grid) Consortium with IBM, an innovative international partnership for computing research and workforce development, with 11 member institutions in the US, Puerto Rico, Mexico, Spain and Argentina.

Transformations for Rapid Prototyping of Time-Critical Applications

Shi-Kuo Chang[1], Zhoulan Zhang[1], Colin J. Ihrig[1]
Paolo Maresca[2] and Valentina Ternelli[2]
[1]Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260 USA
chang@cs.pitt.edu
and
[2]Department of Computer Engineering
University of Naples
Naples, Italy
paomares@unina.it

Abstract: *Application software nowadays tends to be more intelligent to perform actions autonomously, and the development of such software tends to have shorter turn around time. In our approach for designing distributed intelligence systems, each object called a Tele-Action Object (TAO) is enhanced by an index cell (IC). Objects enhanced by index cells can perform actions by themselves. Therefore intelligence is distributed to these tele-action objects. An IC system is an active index consisting of a network of index cells that embodies a lot of distributed knowledge to enhance the overall intelligence of the application software. In the rapid prototyping of time-critical applications, intelligent application software can be generated by combining the technologies of IC cards, IC system (active index), relational mining and time management to transform IC card specifications into executable codes, thus accelerating and simplifying the development of time-critical applications. The Methodology for this transformation approach is described in detail in this paper.*

1. Introduction

Recent advances in communications technology, web-based applications and service oriented architecture have stimulated the need for application software development with shorter and shorter turn-around time. Nowadays customers require application software to possess the following characteristics: (a) it can integrate web services, components and legacy software into a functioning system; (b) it must be operational before deadline and remain operational until expiration time; and (c) it must satisfy user-specified timing constraints. These characteristics indicate that application software should be intelligent and capable of performing autonomous actions.

In our approach for the rapid prototyping of distributed intelligence systems, each object is called a Tele-Action Object (TAO), which is a multimedia object with associated hyper-graph structure and knowledge structure [4]. A Tele-Action Object can be as simple as a single piece of information without connection or relation to any other objects. Or we can combine several TAOs in certain connections into a new complex TAO and/or add certain knowledge to a TAO. Basically the TAO is further refined as two parts (G, K): hyper-graph G and knowledge K. For a TAO the hyper-graph G is used to describe the connections and relations between the sub-TAOs within it. The knowledge K is used to describe the actions.

The private knowledge specific to that object is enhanced by an index cell (IC). Index cells [2] behave like agents; however there can be numerous index cells. Objects enhanced by index cells can perform actions by themselves. Therefore intelligence is distributed to these tele-

action objects. Objects may also contain multimedia data. An IC system is an active index [2, 3] consisting of a network of index cells that embodies a lot of distributed knowledge to enhance the overall intelligence of the application software.

Based upon index cells and tele-action objects, in the rapid prototyping of time-critical applications we propose an approach by combining the following technologies:

– IC cards,
– IC system (active index),
– Relational Mining for appropriate web services,
– Time Management Techniques.

The IC cards enable the visual specification of an application. It captures the *interaction patterns* and *timing constraints*. The interaction patterns depict how the objects interact with each other, and the timing constraints indicate how much time the objects have for action before deadline. The interaction patterns lead to relational graphs specification and protocols, finally the transformation into IC system (active index). The timing constraints lead to time analysis based upon Petri net. The intelligence of ICs in IC system enables the ICs to automatically locate the agents they want to communicate with, and this relational mining mechanism can be used to discover appropriate web services. In component-based software engineering, "gap fulfillment" of the right components uses a similar approach.

The rapid prototyping environment and tools are illustrated by Figure 1.1. The user/developer first creates and edits the IC cards using the IC Card Management System, which generates an XML specification XMLicc. Next the user/ developer can design an IC system based upon the user requirements specified by XMLicc using the Multimedia Knowledge Eclipse Environment, which in turn produces another XML specification XMLicx. The IC Software Engineering Environment accepts the specification of the IC system XMLicx, compiles it into executable code, runs the code and produces runtime snapshots for tracing the execution. Since traceability is maintained by the rapid prototyping environment, the user/developer can go back to change the requirements by modifying the IC cards, redesign the IC system, and test it again. Using the above tools the rapid prototyping and development of time-critical applications becomes easier and more effective.

The paper is organized as follows. In Section 2 we present the IC card structure and its XMLicc schema. In Section 3 the visual editor for the IC system, MKEE, is described. In Section 4 we show where the XMLicc schema and XMLicx schema differ and how to transform one schema into the other. The compilation of the XMLicx of an IC system into codes is explained in Section 5. Section 6 describes the IC software engineering environment. In Section 7 we discuss further research topics.

2. IC Card and its XML schema

An IC card is the user's or developer's specification of an active object or an agent [5]. An active object usually interacts with other active objects according to certain interaction patterns. There are six basic interaction patterns – *quiet* (meaning this active object has no tasks and has no interactions with other active objects), *by-myself-no-interaction*, *by-myself-with-interaction*, *by-others-no-interaction*, *by-others-with-interaction*, and *mixed* (meaning both active objects have to do tasks and they have interactions). Figure 2.1 shows an example of defining an active object using IC card [5], in which interaction pattern and timing constraint are included.

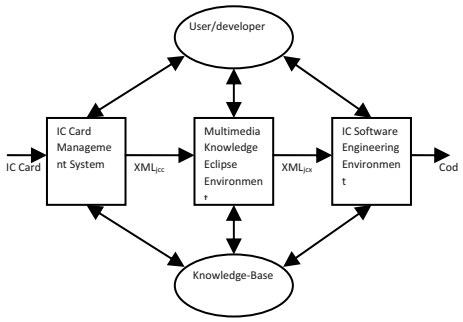


Figure 1.1. Tools for the rapid prototyping environment of time-critical applications.

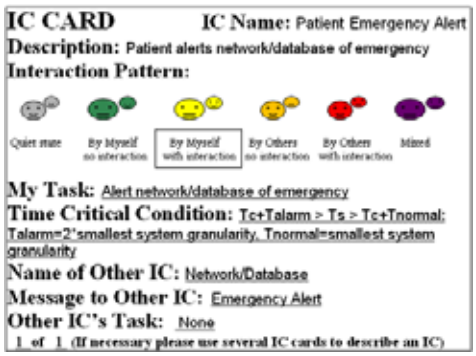


Figure 2.1.An IC card with interaction patterns.

The IC Card Management System enables efficient editing, organization, and management of IC cards. It maintains a list of *icCardEntry*, which is a collection of *icCard*. Each *icCardEntry* has icEntryName, along with EntryId indicating which group an *icCard* belongs to.

For each *icCard*, it includes the following attributes:

– icName: name of the IC
– icId: id of the IC
– icDescription: description for the IC
– icIntPattern: how current IC interacts with another IC, can be one of the six patterns
– icMyTask: task of the current IC
– icTimeCriticalCondition: the timing constraints imposed on the IC
– icNumberTotal: N, the total number of IC cards to describe current IC
– icNumberCurrent: i, the ith IC card (if N IC cards are used to describe the IC)

icCard also has a sub-element *icOther,* which keeps the needed information to interact with another IC. It includes such attributes:

- otherId: the id of the other IC which the current IC will communicate with
- icOtherName: the name of the other IC
- icOtherMessage: the message sent to the other IC
- icOtherTask: the other IC's task

3. The Visual Editor for IC System

The MKEE (Multimedia Knowledge Eclipse Environment, can be found at http://eclipse.dis.unina.it/MkeeSite/) is an Eclipse Development Environment that provides the modeling of multimedia applications and the sharing of knowledge owned by multimedia objects. A multimedia application can be modelled in terms of intelligent objects, i.e., the TAOs [4], which are tele-action objects related by hyper-graph G and enhanced by knowledge K. Every object reacts differently depending on the input that it receives from the outside. The mechanism of answering to the stimuli can be realized by associating a private knowledge to the TAO through the Index Cells [2]. Therefore the application software constructed using this approach supports both the static description of the multimedia application in terms of TAO objects, and the definition of Index Cells net representing the dynamics structure.

A typical IC system is made up of a series of interacting Index Cells, which communicate with each other through message passing. A typical Index Cell has such structure as shown in Figure 3.1:

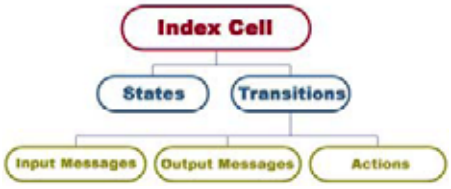


Figure 3.1. The Index Cell Structure.

The Index Cell is a particular Finite State Machine which accepts Input messages, executes operations and sends one or more output messages to one or more IC or to external environment. The amount and type (or types) of IC depends on the state and Input Messages. It is a Mealy model machine and, according to the problem domain, could be deterministic or non-deterministic, but as theory states, any ND-FSM could be transformed in a deterministic FSM.

An example of the visual specification of an IC system based on the healthcare application is illustrated in Figure 3.2. In this IC system, *Camera* captures patient's images and *Sensor* captures patient's health conditions such as blood pressure and temperature. A disabled

patient cannot make alert request by himself/herself, so an alert is initiated by *Sensor* and/or *Camera*. *Emergency Alert* forwards the alert information to *Hospital Response* when any parameter's threshold is reached. Then both the *Doctor* and *Nurse* will be informed. *Nurse* will be dispatched to assist the disabled patient if necessary, and *Nurse* can communicate with *Doctor*.

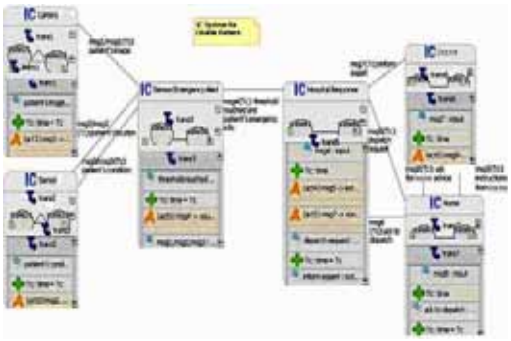


Figure 3.2. A Disabled Patient IC System.

MKEE has the advantage to be a hardware/software platform independent and enable designer to speedily generate application. These advantages are very important in the healthcare environment where a lot of hardware devices and interface to manage there exists. The MKEE tool generates XMLicx specifications of the IC system. Therefore it can be used to serve as the front end for the IC system compiler.

4. Mapping between IC Cards and MKEE IC System

In MKEE the Index Cell is a little different from the IC depicted by IC card. While mapping the same name and id from IC card to Index Cell, Index Cell provides more details such as states and transitions than IC card does. Although the visual representations of IC card Management System and the MKEE IC System are different, they both can be represented by XML, which facilitates the sharing of data across different information systems. Figure 4.1 shows the tree-structured XML schemas used in IC card and IC system.

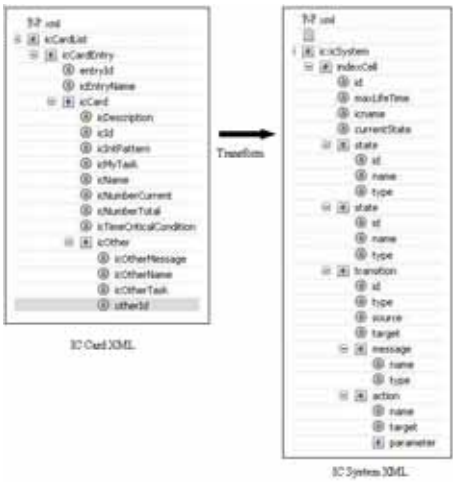


Figure4.1. IC Card XML and IC System XML

The mapping between IC card in IC Card Management System and Index Cell in MKEE IC system is shown in Table 4.1. The matching shows that the two specifications are indeed compatible.

Table 4.1. Map IC Card to MKEE IC System

IC Card Management System	MKEE IC System
icCardType.icId	IndexCell.id
icCardType.icName	IndexCell.name
icOtherType.icOtherMessage	Message.name (The message sent to target IC)
icOtherType.icOtherName	IndexCell.name of other IC (The target IC name of output message)
icCardType.icMyTask/ icOtherType.icOtherTask/ icCardType.IntPattern	Relate to the action target in transition (external/source)
icCardType. icTimeCriticalCondition	Parameter.dataValue (Time parameter in messages)

Specifically for the timing constraints, users can indicate the time during which a task should be done in an IC card's TimeCriticalCondition field. In an IC system, similar timing constraints can be imposed through the parameter of message in transition.

Since both IC Card Management System and MKEE are built on Ecore models, we can use IBM Model Transformation Framework (MTF) to implement partial transformations between the two models. MTF provides an extensible rules language that can be used to define what the

transformation should accomplish, and a transformation engine which can interpret the rules in order to perform the transformation. MTF works on models described by a compatible meta-model in order to express the correspondences in a consistent way. The output of MTF transformation is a set of mappings that relate the objects of two models. The user can specify the mapping as a *relation* that defines the type of mapping that will apply to model class instances.

5. Compiling IC Specifications

Modern compilers are typically made up of five conceptual phases. The first three phases, lexical analysis, syntax analysis, and semantic analysis, are grouped together to form the compiler's front end. The fourth conceptual phase is code optimization and is referred to as the middle end. The final phase, dubbed the back end, is the code generator.

The front end is responsible for recognizing validity, or lack thereof, of source language. It also shapes the source language into an intermediate representation (IR), typically an abstract syntax tree (AST) for the middle and back ends. In the case of the IC system, the MKEE visual editor creates the AST. The AST is then analyzed for proper semantics by the compiler. Upon completion of semantic analysis, the task of the front end is complete.

The middle end of the compiler analyzes and transforms the IR through optimizations, so as to improve code quality such as reduced execution time. The IC system compiler does not currently perform any optimizations.

The back end generates the final output in the target language. Contrary to the front end, the back end is source language independent and target language dependent. The IC system compiler generates Java classes, which can be used in cross-platform web applications.

5.1. Input to the IC System Compiler

The MKEE tool is an Eclipse visual editor for specifying IC system. It outputs an XMLicx file describing an IC system. XML documents, due to their inherit tree structure, provide an obvious mapping to an AST in a typical compiler. Once the XML file is created, it can be processed by the IC system compiler. The compiler reads in the XML file, and creates an internal IR. The MKEE tool outputs valid XML files. However, the XML may not be semantically correct for code generation. Semantic analysis is performed on the IR, and, pending valid semantics, output as a set of Java classes. The Java classes can in turn be used in web based *Java Server Pages* (JSP) applications. When a JSP page is accessed, a web page is dynamically created by compiling the Java classes used by the page. The use of server side computation and the Java programming language allows for platform independence and rapid development of web applications.

5.2. Semantic Analysis

The top level of the IC system is a collection of Index Cells. Index Cells are composed of States and Transitions. For type checking purposes, all immediate children of an Index Cell must be of type State or Transition. States themselves must have a type of internal, entering, or ending. A value of entering indicates that this is a start state. All Index Cells must have exactly one State of type entering. There can be multiple ending and internal states. Each Index Cell must also have a maximum lifetime value. Index Cell's which do not persist forever will have a numeric value specifying the lifetime in milliseconds. Once the length of the lifetime has expired, the Index Cell will die.

Transitions are hierarchal objects composed of Actions, Input Messages, and Output Messages. Transitions have a source state and a target state. Both the source state and the target state must be resolvable to valid State structures. Transitions also must have a type of boundary or internal. Internal transitions occur between states in the same Index Cell. Boundary transitions occur between states in different Index Cells and are used to pass messages. Internal transitions

areused to drive the progression between States in an Index Cell.

Messages and Actions are composed of Parameters. Parameters must have a data type and data value, as well as a name property. Output Messages have a Target Index Cell that must be resolvable to a valid Index Cell. Actions have simple semantics, requiring only a name.

5.3. Code Generation

The code generator of the IC system compiler creates a set of Java classes that can be used in JSP applications. The code generator is based on a set of transformation rules. The following example illustrates the transformation from MKEE formatted XML to Java code. It is worth mentioning that names in MKEE are entered by the user. The user-provided names are transformed to ensure that they are valid identifiers in the Java language. A simple Index Cell taken from the previously described disabled patient IC system is shown in Figure 5.1. The visualization of the Index Cell will result in the MKEE XML in Listing 5.1. The MKEE XML is passed to the compiler. The compiler will apply a set of transformation rules to the XML resulting in Java source code. Table 5.1 contains a non-exhaustive listing of transformation rules used in the compiler. Transforming most of the XML is a relatively straightforward process. Transitions provide the most challenging transformation process. The Transitions along with the States are transformed into the Index Cell's transition function. The Java implementation of the transition function for the example Index Cell is shown in Listing 5.2.

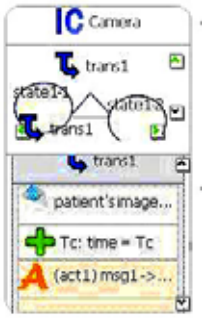


Figure 5.1. Visualization of an Index Cell.

```
<indexCell
currentState="//@icSystem/@indexCell.0/@state.0" id="ic1"
maxLifeTime="infinity" name="Camera">
   <state name="state1-1"/>
   <state name="state1-2"/>
   <transition id="trans1"
source="//@icSystem/@indexCell.0/@state.0"
target="//@icSystem/@indexCell.0/@state.1">
     <message xsi:type="ic:OutputMessage" id="msg1"
name="patient's image">
       <targetIC>ic3</targetIC>
       <parameter dataType="time" dataValue="Tc"
name="Tc"/>
     </message>
     <action body="collect image info periodically" id="act1"
name="msg1" target="source"/>
   </transition>
</indexCell>
```

Listing 5.1. Specification of an Index Cell.

Table 5.1. Example Transformation Rules.

MKEE XML		Java Transformation
<indexCell id="α">	→	public class α extends IndexCell
<indexCell currentState="β">	→	This.currentState = β;
<parameter dataType="τ"/>	→	new τ()
<action id="ρ">	→	private void ρ()
<targetIC>ψ</targetIC>	→	Ψ
<message id="ω">	→	new ω ("α", "ψ", τ)

```
public void transition() {
  switch( this.currentState )
  {
    case icSystemindexCell0state0:
      changeState( icSystemindexCell0state1 );
    break;
    case icSystemindexCell0state1:
      if ( this.previousState ==
           icSystemindexCell0state0 ) {
      act1();  postMessage( new msg1( "ic1", "ic3",
      new Object() ) );
      }
      else stateError();
    break;
    default:
      stateError();
  }
}
```

Listing 5.2. Example Transition Function.

In the target code, each Index Cell is implemented as a separate Java class. Each Index Cell class is a thread, in order to allow multiple Index Cells to run concurrently. The States of an Index Cell are implemented as a finite state machine based on the Transitions between them. Messages are implemented as objects which are passed between Index Cells.

The Messages' Parameters are implemented as the objects' data fields with corresponding inspector functions. Actions are implemented as function calls. An Action's Parameters correspond to the arguments to the function. Each Index Cell transitions through its state machine until its lifetime expires. On state transitions, any corresponding output messages and actions are executed. If the next state expects an Input Message, the thread will sleep until it receives any messages it is expecting. Finally, the next transition will be taken.

The code generator can be enhanced by relational mining as follows. If through the relational mining technique an appropriate web service is found to perform certain functions, then the code generator will only produce the Java class to invoke the available web service.

6. The IC Software Engineering Environment

The IC Software Engineering Environment (ICSEE) is an environment for compiling and generating Java codes from MKEE XML output. Figure 6.1 illustrates the interface of ICSEE. The ICSEE main screen shows the various stages including *UploadXML*, *Parsing*, *Generate*, *Compile*, *Load* and *Instantiate*, as well as the options of *Save*, *Delete*, *Logout*, and *ReadMe*. We will also use the disabled patient example here. Figure 6.1 illustrates the results after *UploadXML*, showing the various Index Cells.

After uploading the disabled patient XML which is generated by MKEE, click the "*Parsing*" button. Then all the Index Cells including Camera, Sensor, Sensor Emergency Alert, Hospital Response, Expert, and Nurse will be parsed and displayed in "Index Card Information" section. The Java source code will be generated at the server side when user clicks "*Generate*". The server will start compiling the generated Java source code, and the compiled classes will be saved. When clicking "*Load*", the generated classes will be loaded into JVM; when clicking "*Instantiate*", the created instances will be displayed in the "Index Cell Instances" section.

Using ICSEE the user/developer can easily test and trace the rapid prototyping process. The generated Java code also lays the foundation for further developing application software.

7. Discussion

The transformation approach for rapid prototyping of intelligent software applications is described in this paper. The transformation from IC systems into codes enables the user/developer to explore the design space based upon different IC systems. However the transformation from IC cards to IC systems is still at best a manual process. Our next objective should be to (partially) automate this transformation.

There are different ways to realize the transformation from IC cards to IC system: developing syntactic transformation on the XML documents, or developing graph transformation algorithms to produce a target graph from the initial graph, or using some AI techniques such as rule-based approach. Following the first approach, now we can find the matching between IC cards and IC system and define the mapping rules between the two models. However the IC cards represent an initial specification without details about interaction protocols, timing considerations and protocols. The mapping rules are incompletely defined. Therefore the other approaches are still necessary and require further investigation.

Acknowledgements:

Giuseppe Marco Scarfogliero and Lorenzo Sorrentino implemented the initial version of MKEE, the IC System Visual Editor. Dan Li implemented the software engineering environment (ICSEE) for the IC System Complier. Robert Zaremba provided the details on the implementation of remote IC subsystems. Their contributions are acknowledged.

References:

[1] Kent Beck and Ward Cunningham, "A Laboratory For Teaching Object-Oriented Thinking", Proceedings of 1989 ACM OOPSLA Conference on Object-Oriented Programming, 1989, 1-6.

[2] Shi-Kuo Chang, "Towards a Theory of Active Index", Journal of Visual Languages and Computing, Vol. 6, No. 1, March 1995, 101-118.

[3] Shi-Kuo Chang, D. Graupe, K. Hasegawa and H. Kordylewski, "An Active Medical Information System using Active Index and Artificial Neural Network", in Advances in

Medical Image Databases, (S. Wong, ed.), Kluwer, 1998, 225-249.

[4] H. Chang, S. K. Chang, T. Hou and A. Hsu, "The Management and Applications of Tele-Action Objects" , ACM Journal of Multimedia Systems, Springer Verlag, Volume 3, Issue 5-6, 1995, 204-216.

[5] Shi-Kuo Chang, Perry Rajnovic and Mark Zalar, “IC Card: Visual Specification for Rapid Prototyping of Time-Critical Applications”, International Journal of Software Engineering and Knowledge Engineering, Vol. 17, No. 5, October 2007.

[6] P. Maresca, S.K. Chang, M. Pesce, "Application of Active Index to the Management of E-Learning", Rivista della societ` italiana di e-learning Rivista Si-el, Vol. 3, No. 2, 2006, November 2006, 331-341.

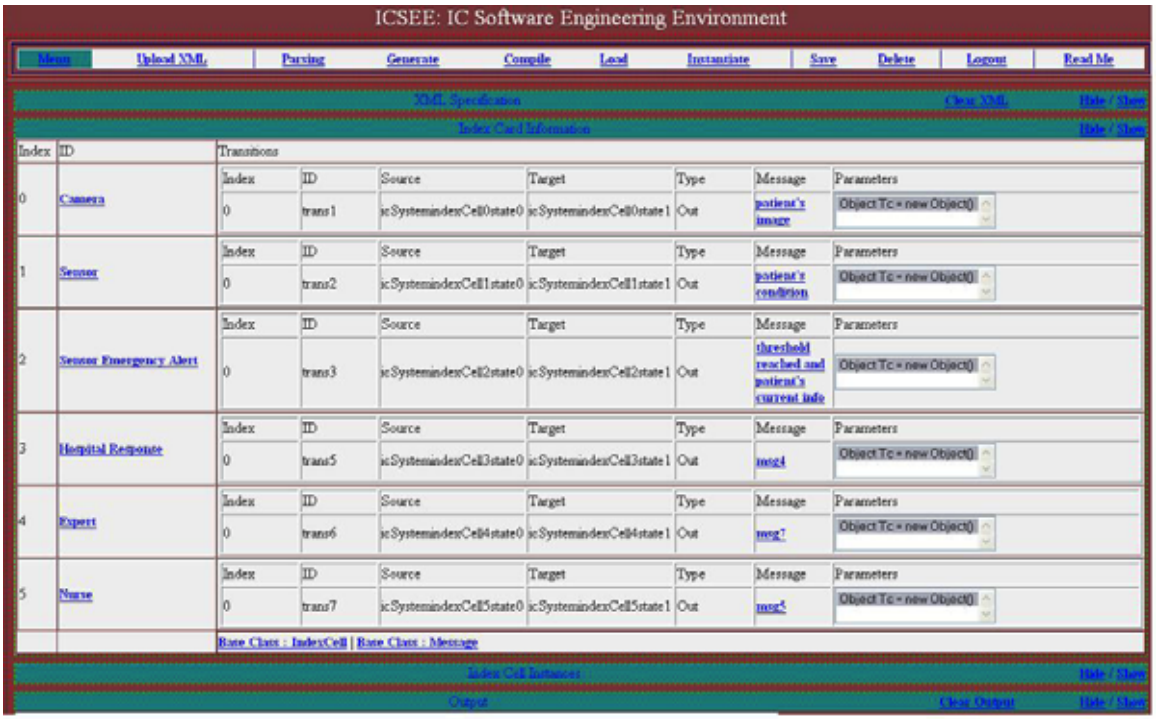


Figure 6.1. Results after *UploadXML*.

Case Study: Applying Business Process Management Systems

Gregor Scheithauer* and Guido Wirtz
Distributed and Mobile Systems Group, University of Bamberg
Feldkirchenstraße 21, 96052 Bamberg, Germany
gregor.scheithauer@gmail.com, guido.wirtz@uni-bamberg.de

Abstract

Business Process Management Systems aim to support the Business Process Management paradigm and to ease legacy application integration. However, do such systems really meet real-world requirements? This paper introduces and discusses a set of criteria which are important for business process management systems and applies these criteria in comparing tools from three important vendors, namely IDS Scheer*,* Oracle *and* Intalio *based on a real-world case study.*

Keywords: BPMS, usability criteria, Webservices

1. Introduction

Business Process Management Systems (BPMS) [7] are sets of tools to support the Business Process Management (BPM) life-cycle [5] that are either offered by one vendor, or multiple vendors offer parts of a BPMS. Smith [6] sees a list of key advantages in using a modern BPMS: it bridges heterogenous application environments, includes human activity by incorporating workflow, allows web service orchestration, provides the opportunity to customize the whole process for specific customers and partners, offers an integrated user interface through a single portal and back-end integration, and monitors process instances. Rather than introducing new technology or replacing existing business applications, BPMS integrate existing technologies and existing applications in a process-oriented fashion. Based on this notion of BPMS, Smith and Fingar [7] describe requirements for a BPMS as follows: a BPMS should be able to support modeling, deploying, and monitoring business processes, as well as to support integration of heterogeneous processes, automatization, and collaboration.

Table 1 depicts which BPMS tools support what step in the BPM life cycle. Business process design includes process documentation with a process notation, such as Event-driven Process Chain (EPC) [3] notation and Business Process Modeling Notation (BPMN) [9]. Configuration includes the transformation [2] from process models into formal languages such as the Business Process Execution Language (BPEL) [1]. Integration facilitates better reuse of existing applications. BPMS allows easy deployment of configured process models, and to execute them.

*The first author is funded by means of the German Federal Ministry of Economy and Technology under the promotional reference 01MQ07012.

This paper summarizes the results of a case study to find out if prominent existing BPMS meet real-world expectations. Based on a detailed list of evaluation criteria covering all steps relevant for BPMS (section 2), a real-life scenario is used to evaluate two different BPMS - a multi-vendor system based on tools from *IDS Scheer* and *Oracle* as well as a single vendor system provided by *Intalio* (section 3). The results are summarized in table 2. A more detailed description of the scenario and the case study implementation as well as a more in-depth discussion of the results is documented in [4].

2. Evaluation Criteria

The criteria used to evaluate BPMS tools take a holistic view on the entire process. The 23 criteria are clustered into three layers which are introduced in [5]: questions 1–9 cover the business layer, questions 10–19 address the integration layer, and questions 20–23 address the execution layer. The questions represent real-world requirements originating from an industry project.

1. *What kind of people are involved during design and improvement in the BPM life cycle*. These steps need to be business driven, and flexible, thus, people who manage business processes, need to be in the position to express their understanding of business, without technically founded limitations.
2. *Standard or proprietary design notation* points out if the process design notation in question was standardized by a group such as OMG or OASIS, or if it is a vendor specific format. Moreover, does the standard cover the graphical elements and the persistence of the notation? By using a standard notation, it is easy to switch process design tools or exchange process diagrams between different process design tools.
3. *Industry acceptance* shows if a process design notation is widely used in industry. Established notations are more likely to provide supporting technologies and middle-ware. In addition, if a design notation is widespread, it might undergo further and constant improvements.
4. *Completeness* of process design notations denotes the expressive power of a notation (cf [8]). Business

Table 1. BPM tools used in the process life cycle

		Design	Configuration	Integration	Deployment	Execution
IDS Scheer (ARIS)	SOA Architect		X	X		
	Business Architect	X				
Oracle	Process Manager					X
	BPEL Designer		X	X	X	
Intalio	Process Server					X
	Process Designer	X	X	X	X	

analysts need elements to express business tasks, business objects, and business partners. Missing elements result in complex process diagrams emulating missing constructs, which are difficult to maintain.

5. *Data management*. indicates the possibility to design business objects with the process design tool. Business objects make the process diagram semantically richer and better to understand for process stake-holders.
6. *Is a methodology behind process design notation*. A methodology covers the semantics of the notation and reduces the complexity of business process design via guidelines how to use and how to combine the elements of the notation.
7. *Does the design tool support the full design notation in its recent version.* The more a tool supports a design notation standard, the greater the ability to exchange process diagrams.
8. *Diagram repository* states if the process design tool accesses diagrams from a shared repository or from a local machine. A process repository has the advantage that more people are allowed to access processes, thus, processes are viewed and re-viewed by more people, which, to an extend, improve process diagrams.
9. *Process version control* shows if a process design tool contains or has access to version control. Next to a shared repository, this is a very useful tool for maintaining process diagrams. Business analysts are able to roll back to a prior version of the process, if necessary, or browse the evolution of a process for a better understanding of the meaning behind the current version.
10. *What kind of people are involved in the configuration and the integration step of the life cycle.* Process diagrams should not be altered much to be executed. No change in business logic should be needed, but a technical mapping is required. People on this level must not be faced with the complexity of business logic.
11. *Compatibility of design notation and execution language* refers to what extent the design notation is transformable into the execution language. There are two main reasons for incompatible languages: (i) languages are either block-oriented or graph oriented or, (ii) languages may support different concepts and use richer semantics.
12. *Standard or proprietary execution language* points out if the execution language in question was standardized, or if it is vendor specific. This covers the language and the persistence of the language. Using a standard language eases switching execution engines or exchanging process configurations between different engines.
13. *Industry acceptance* shows if an execution language is widely used in industry. Besides the importance to use standards, it is necessary to find supporting technologies and middle-ware to support execution languages.
14. *Message type management*. Is it possible to design or even import message types with the configuration tool? Next to configure the flow of business tasks between applications, departments and companies, it is necessary to define message types. These types may be imported from service definitions, database table definitions or class definitions from a programming language. Otherwise, they might be defined with the configuration tool.
15. *Configuration complexity* measures how many tools are needed for a successful process configuration. Besides an integrated configuration tool, it may necessary to apply configuration to other middle-ware before deployment is possible. The more tools and middle-ware need to be configured, the higher the complexity.
16. *Is process configuration part of a shared repository*. This criteria points out if the configuration tool accesses process configurations from a shared repository or from a local machine. The former has the advantage that more people access the configuration, thus process configuration might be adapted by many people.
17. *Is the process configuration attached to the process diagram*. If there is a well-defined link between a process diagram and the process configuration, changing the diagram as well as the configuration consistently becomes much easier.
18. *Is process configuration bound to one execution platform* refers to the *vendor lock* issue. This is the case, if process configurations are only be executed on the platform which the process diagram was configured with. This may happen if execution engines do not support standards or industry accepted execution languages. A vendor lock makes it difficult to switch between different execution engines.
19. *Legacy applications integration* explains what kind of applications and their services may be integrated. However, middle-ware technology makes it possible to integrate those application as services.
20. *What kind of people are involved in the deployment step*. System analysts should be qualified to accomplish this task. If other than the system analyst needs to be involved, process deployment is a too complex step.

21. *Deployment tool integration* tells whether a deployment tool is integrated into an IDE or not. Users do not need different tools, the acceptance of the user is higher and users already know how the tools behave.
22. *Deployment complexity* measures how many tools are needed for a successful process deployment. Next to an integrated deployment tool, it may be necessary to deploy to more than one execution engine. The more deployment steps are required, the higher is the complexity for process deployment.
23. *Process version control*. This refers to what will happen if instances of a process are running and a new version of that process will be deployed. There are four possibilities. Firstly, all instances are stopped and deleted. The new process will be deployed. Secondly, a deployment of a new version is refused, when instances of that process are still running. Thirdly, the tool tries to merge running instances with the new process definition. If a merge is possible, the new version will be deployed, otherwise the deployment will be refused. Running instances may run until they terminate. New instances are based on the new version of the process. The old version of that process will be archived when every instance has been terminated.

These criteria are used to evaluate the different tools when realizing the example process that is introduced next.

3. Case Study and Results

Two companies are involved in the case study: *Shade Tree Garage* (STG), a garage shop in New Jersey, repairs cars for nearly all makes of cars wehereas the *SPC* company manufactures car spare parts and distributes them to garage shops. Prices for spare parts are not fixed and change on a daily basis. Shade Tree Garage wants to minimize its stocking costs and to maximize planning reliability. SPC identifies this demand as a selling proposition, and intends to offer a *Garage Shop Information System* (GSIS) to garage shops.

The business process, which is shown in figure 1, offers price information and quantity information for spare parts to garage shops. On the business level, the following business tasks are identified: (1) *Request spare part information* on the garage shop side, (2) *Receive spare part information request*, (3) *Get price information for spare part*, (4) *Get quantity information for spare part*, and (5) *Send spare part information* on the SPC side. The business objects include (1) *unique ID for spare parts*, (2) *Price*, and (3) *Quantity*. On the service level, two services are needed: (1) *PriceService*, and (2) *QuantityService*. Both services are available as web services and provide a WDSL file. The appropriate message exchange pattern between SPC and garage shops is a Request-Response pattern. To access the GSIS, the *GSISRequestMessage* is used which contains a placeholder for a spare part ID. Spare part information is received by the *GSISResponseMessage* which contains a placeholder for price and quantity information.

The case study comprises an end-to-end business process that contains reasonable business logic and has relevance in today's business. Moreover, it spans more than one company's department and more than a single application. Hence, it is suitable to check technical capabilities and business to business integration issues. The results of applying our criteria when implementation GSIS using two different BPMS are summarized in table 2; for a detailed discussion refer to [4].

4. Outlook

Future work has to include better integration of different tools into a BPMS. Further adoption and improvement of standards, such as BPEL, and WSDL might tackle this issue. Tool providers must enhance tool functionality and better separate the roles in the life cycle. Moreover, process design notations must advance in the direction of BPMS, that is, that business processes are intended to be supported by webservices. Lastly, as business processes become executable and traceable by means of process portals, BPMS should permit the monitoring of important data and processing of this data. Business analysts might even model Key Performance Indicators (KPI) with a process design notation and get processed results for process instances on those indicators. This allows to identify bottlenecks and shortages in business processes.

Only after lots of ambitious efforts and their successful completion over the next years, BPMS will become easier to use during the entire life cycle of business processes.

References

[1] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, and C. K. Liu. Specification: Business Process Execution Language for Web Services version 2.0. Tech. rep. 2.0, OASIS, January 2007.

[2] C. Ouyang, van der Aalst, M. Dumas, ter Hofstede, and A. H. M. From business process models to process-oriented software systems: The BPMN to BPEL way, October 2006.

[3] A.-W. Scheer and M. Nuettgens. Architecture and Reference Models for Business Process Management. *Lecture Notes in Computer Science*, 1806 / 2000:376–389, 2000.

[4] G. Scheithauer and G. Wirtz. Applying Business Process Management Systems – a Case Study. Bamberger Beiträge zur Wirtschaftsinformatik 76, Bamberg University, May 2008. ISSN 0937-3349.

[5] G. Scheithauer, G. Wirtz, and C. Toklu. Bridging the semantic gap between process documentation and process execution. In *The 2008 International Conference on Software Engineering and Knowledge Engineering (SEKE'08)*, 2008.

[6] H. Smith. The Emergence of Business Process Management. *Information & Software Technology*, 45(15):1065–1069, December 2003.

[7] H. Smith and P. Fingar. *Business Process Management, the third wave*. Meghan-Kiffer Press, first edition edition, January 2003.

[8] R. Weber. Ontological foundations of information systems. *Coopers & Lybrand: Accounting Association of Australia and New Zealand, Melbourne*, 1997.

[9] S. A. White. Specification: Business Process Modeling Notation Specification, February 2006.

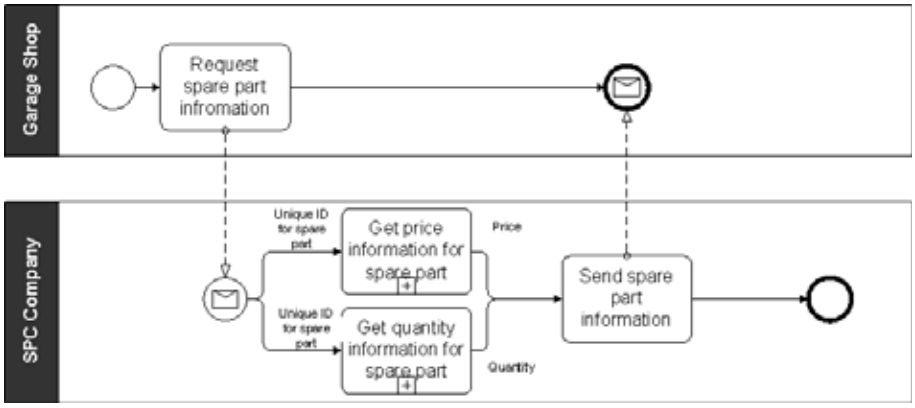


Figure 1. Case Study Process

Table 2. Evaluation: 1-9 – Business Level / 10-19 – Integration Level / 20-23 – Execution Level

	ARIS Business Architect	**Intalio BPMN Designer**
1.	Process owner, business analyst, process participant	Business analyst
2.	EPC; not a standard, was published in 1992, XML based, thus open.	BPMN is an OMG standard, may replace UML activity diagram, only standard for graphical elements, not easy exchanged.
3.	EPC is accepted in the industry for documenting and communicating processes within a company. Recently, SAP and Oracle use ARIS tools to enable business analysts to interact with middleware.	BPMN is seen as a workflow definition language, since it is very rich in graphical elements. Business people hesitate to use it since it is so rich at technical elements. Tool support is available.
4.	Complete	BPMN in general.
5.	Possible	Possible with limitations
6.	Architecture Integrated Information Systems (ARIS)	Specification with instructions how to use the notation
7.	Full support	Limited support
8.	Global or local diagram repository	No repository
9.	No version control	No version control
	ARIS SOA Architect, Oracle BPEL Designer	**Intalio BPMN Designer**
10.	System analyst, software developers	Business analyst, system analyst, software developer
11.	EPC → BPEL 1.1 Poorer to richer semantic translation	BPMN → BPEL 2.0 Different semantics
12.	BPEL 1.1, Specification from IBM & Microsoft	BPEL 2.0, Specification from OASIS
13.	Accepted language, great tool support, missing concepts, such as human people integration	BPEL 2.0 Specification not yet adopted, no other tool support
14.	Complete	Complete
15.	Configuration with two major tools necessary	Configuration with one tool
16.	Global or local service repository	Local service repository
17.	Process diagram and process configuration are linked, though only the configuration done in the SOA Architect. It is possible to synchronize changes.	Process diagram and process configuration are attached. Changes to either process diagram or process configuration affects the other.
18.	No vendor lock	Since BPEL 2.0 is not widespread, process configuration is limited to Intalio's BPMS
19.	Integration through web services	Integration through web services (Connectors for SAP)
	Oracle Process Manager	**Intalio Process Server**
20.	System Analyst Software Developer	System Analyst Software Developer
21.	Integrated into the BPEL Designer	Integrated into the BPMN Designer
22.	BPEL Designer and Process Manager	BPMN Designer and Process Server
23.	Parallel	Newer versions overwrite older versions

Verification of optimization algorithms: a case study of a quadratic assignment problem solver

Tsong Yueh Chen[*], Huimin Lin[†], Robert Merkel & Daoming Wang

Abstract

It is often difficult to verify the solutions of computationally intensive mathematical optimization problems. Metamorphic testing is a technique to verify software test output even when a complete testing oracle is not present. We apply metamorphic testing to a classic optimization problem, the quadratic assignment problem (QAP). A number of metamorphic relations for the QAP are described in detail, and their effectiveness in "killing" mutated versions of an exact QAP solver is compared. We show that metamorphic testing can be effectively applied to the QAP in the absence of an oracle, and discuss the implications for the testing of solvers for other hard optimization problems.

1 Introduction

In software testing, an "oracle" is a means by which it is determined whether the output of a test case meets the software's specification. In some cases, a convenient oracle is readily available; however, there are many situations where the oracle's evaluation is manually conducted, a slow and error-prone process. For some applications - a notable example is scientific simulation - even this may not be possible.

Metamorphic testing is a testing technique designed to allow the checking of test output where there is no oracle, or it is too expensive to verify against the oracle [1]. In this paper, we investigate the use of metamorphic testing on a class of problems where oracles are difficult to find - mathematical optimization problems. A classic example of this class, the quadratic assignment problem, serves as the subject of our case study.

1.1 The Quadratic Assignment Problem

Mathematical optimization problems, in general, involve finding the maxima or minima of functions, usually taking into account constraints of the function parameters. Many practically important classes of optimization problems are NP-hard[1], or even harder. In practice, given that we need an actual solution rather than simply knowing the existence of one, this means that verifying that a solution is indeed optimal can be very difficult.

The Quadratic Assignment Problem (QAP) is a classic example of this kind of difficult operations research problem. Informally, the problem can be viewed as "the assignment of facilities to locations" [2]. For example, consider a company which intends to build k factories denoted as $f_1, f_2, \ldots, f_k$. The factories can be constructed in k distinct locations $l_1, l_2, \ldots, l_k$. The output of some factories is used as the input to others. The amount of goods that will move between factories is quantified as the "weight". This weight can be represented as a $k \times k$ matrix w, where $w(i, j)$ represents the weight of the items moving from factory f_i to factory f_j.

The relative expense of moving items around between locations can be represented in another $k \times k$ matrix c, where $c(i, j)$ represents the cost of moving a unit of weight from location l_i to location l_j. In many real-world applications, the cost corresponds to the distance between two locations. Therefore, it is common for $c(i, j)$ and $c(j, i)$ to be the same for all i and j, and thus for the matrix to be symmetric, but this is not compulsory according to the formal definition of the problem. The quadratic assignment problem is to assign factories to locations such that the total transport cost is minimised. More formally [3], the goal is to find a bijection $g : \{f_1 \ldots f_k\} \rightarrow \{l_1 \ldots l_k\}$ such that the following summation is minimised:

$$\sum_{i=1}^{k} \sum_{j=1}^{k} w(i, j)\, c(g(i), g(j)) \tag{1}$$

Like many other optimization problems, the solution g may not be unique (as will be illustrated in section 2.1).

Given that the problem is NP-hard, and the obvious practical applications, there has been a great deal of research to find efficient approximation algorithms for the QAP (see [4] for a survey).

In this paper, however, our interest in the QAP was in demonstrating the detection of faults in an exact QAP solver.

[*]T.Y. Chen and Robert Merkel (corresponding author) are at at Swinburne University of Technology, John St. Hawthorn 3122, Email: {tchen,rmerkel}@swin.edu.au

[†]Huimin Lin and Daoming Wang are at the Institute of Software, Chinese Academy of Sciences, Beijing, China, Email: {lhm,dmwang}@ios.ac.cn

[1] in their decision problem form

Lau [5] provides such a solver, implemented in Java. Primarily intended for educational purposes, this solver meets our needs as a relatively straightforward implementation that could be easily incorporated into a test harness. Lau's code base contains a large number of other methods that implement other optimization algorithms, that are not executed when using the QAP solver. To avoid irrelevant and time-consuming processing of this code in our experimental setup, methods unrelated to the QAP (and thus never executed) were removed from the source code.

1.2 Metamorphic Testing

Metamorphic testing [1] is a method for verifying test output in the absence of an oracle. It is based on the idea of taking pre-existing *source* test cases T, and systematically generating a set of *follow-up* test cases T' based on T. While it may not be known directly whether the program output on individual members of T or T' is correct, T' is constructed in such a way so that certain relations between T, T', and their corresponding outputs, must hold if the software is functioning according to its specification.

To take a simple example, consider that the software under test is designed to compute the sine of an angle θ. Elementary trigonometry tells us that $\sin(-\theta) = -sin(\theta) \forall \theta$. We can use this property as a *metamorphic relation* to derive follow-up test cases and relationships between the outputs for the original and follow-up test cases. Therefore, if we have executed a test case θ_i with output s_{θ_i}, we can derive a follow-up test $-\theta_i$. The output from the execution of the follow-up test case $-\theta_i$ must be equal to $-s_{\theta_i}$ for the metamorphic relationship to hold. If it does not, then a failure in the software under test is revealed.

This metamorphic relation is a simple one-to-one correspondence between a single source test case and a single follow-up test case, but metamorphic relations can be defined between multiple tests. For instance, consider the trigonometric tangent function. The tangent addition formula states that for angles θ and ϕ, $\tan(\theta + \phi) = \frac{\tan\theta + \tan\phi}{1 - \tan\theta\tan\phi}$. To use this metamorphic relation with two source test cases, θ and ϕ, a follow-up test case $(\theta + \phi)$ is constructed and executed, and the metamorphic relation is then checked. Furthermore, while these examples have made use of an equality relationship, this is not obligatory; other relationships can be defined, such as inequalities, greater-than or less-than relationships, or non-numeric relationships such as subsets.

Metamorphic testing has been examined in a number of contexts, including COTS component testing [6], context-sensitive middleware [7], and programs involving symmetries [8]. It has been shown to provide effective testing in the absence of an oracle.

In this paper, we seek to demonstrate the application of metamorphic testing for a solver of a well-known optimization problem.

2 Some Metamorphic Relations for the QAP

For most software under test, there are many valid metamorphic relations that might be used to check software correctness. However, not all of them will be effective in revealing software failures; as a trivial example, if we have a program P that takes some input I (assuming there is no internal state), and returns P(I), clearly P(I) = P(I) for all I. However, this will probably reveal only a very few failures.

We devised a number of metamorphic relations that we hoped would be effective for evaluating the correctness of a QAP solver. The three types of metamorphic relations (six relations in total), were identified quickly, over a few hours. None of the authors had any previous experience in the area of QAP solvers. All the metamorphic relations we constructed, were evaluated experimentally. We did not cherry-pick a few satisfactory metamorphic relations from a much larger, mostly unsatisfactory bunch!

2.1 Relabelling

The labelling of factories and locations in the QAP does not affect the nature of the solutions, only their names. For instance, assume that we have a QAP of size 3 with an optimal assignment of f_1 to l_2 f_2 to l_1 and f_3 to l_3. If we relabel factories f_1, f_2, and f_3 as $f_2\prime$, $f_3\prime$, and $f_1\prime$ respectively, without changing the weights between the relabelled factories, we know that assigning factory $f_2\prime$ to l_1, $f_3\prime$ to l_2, and $f_1\prime$ to l_3 will be optimal for the relabelled QAP.

However, for any given QAP, there may be more than one optimal solution - for a trivial example, consider a QAP where all weights are zero and hence every possible assignment is an optimal solution. However, the QAP solver does not guarantee to return any particular optimal assignment. Therefore, we cannot assume that the optimal solution found to a relabelled problem will be the corresponding relabelling of the optimal assignment found for the original problem. However, all optimal solutions by definition have the same total cost; therefore, any optimal solution to the relabelled problem will have the same total cost as the optimal solution to the original problem.

Consider the QAP denoted by Q_a with weight and distance matrices defined as follows:

$$w_a = \begin{matrix} 0 & 5 & 8 \\ 0 & 0 & 0 \\ 0 & 4 & 0 \end{matrix}, c_a = \begin{matrix} 0 & 7 & 2 \\ 7 & 0 & 10 \\ 2 & 10 & 0 \end{matrix} \quad (2)$$

If we relabel the factories such that $f_1\prime = f_3$, $f_2\prime = f_1$, and $f_3\prime = f_2$, we can obtain a weight matrix $w_a\prime$:

$$w_a\prime = \begin{matrix} 0 & 0 & 4 \\ 8 & 0 & 5 \\ 0 & 0 & 0 \end{matrix} \quad (3)$$

Similarly, if we relabel the locations $l_1\prime = l_2, l_2\prime = l_3, l_3\prime = l_1$, we get a distance matrix $c_a\prime$:

$$c_a\prime = \begin{matrix} 0 & 10 & 7 \\ 10 & 0 & 2 \\ 7 & 2 & 0 \end{matrix} \quad (4)$$

The QAP instance $Q_a\prime$, with weight matrix $w_a\prime$ and distance matrix c_a has an optimal solution with exactly the same cost as Q_a. The QAP instance $Q_a\prime\prime$, with weight matrix w_a and distance matrix $c_a\prime$ will also have an optimal solution with the same cost, as would (though we did not use this) $Q_a\prime\prime\prime$ with $w_a\prime$ and $c_a\prime$.

We implemented this metamorphic relation by developing a tool that randomly permutes the weight of any given problem instance, and then separately permutes the distance of the problem instance, resulting in two follow-up problem instances. The QAP solver is then used to solve both the source problem instance Q_a and the two follow-up problem instances $Q_a\prime$ and $Q_a\prime\prime$. The total cost of all three must be equal.

2.2 Adding a new factory

There is no straightforward way to predict the effect on the optimal solution after an arbitrary addition of a factory and location to an existing QAP. However, if it is possible to ensure that any optimal solution must have the new factory being placed in the new location, calculating the cost of the optimal solution to the expanded problem, is then quite straightforward.

One way to ensure this is to make the new location a very long distance M from all the existing locations (much greater than distances between existing factories), and assign zero weights between the new factory and all existing factories. As discussed earlier, there is the possibility that there are multiple optimal solutions to the original QAP, and therefore there are multiple optimal solutions for the expanded QAP. If this is the case, we cannot be sure that the optimal solution found by the solver for the expanded QAP will simply be the optimal solution found for the original one, with the new factory assigned to the new location. Regardless, any optimal solution must have the new factory in the new location, and hence the total cost must not change.

One potential flaw in such a scheme is that there are still certain obvious types of faults that such a relation will not detect. For instance, consider a bug in the cost calculator such that it always finds a total cost of 0, regardless of the assignment of factories to locations. Therefore, a modified version of this metamorphic relation was sought. Instead of zero weights between the new factory and all existing factories, one existing factory f_i is chosen randomly, and a weight smaller than all existing non-zero weights is assigned between the new factory f_n and f_i, with weights between the new and existing factories other than f_i set to 0. In this case, the new factory will still be assigned to the new location, but the cost of an optimal solution will be increased. If the distance between f_n and f_i is d, and the weight is ϵ, the total cost will increase by $d \cdot \epsilon$. For this relationship to hold, there must be no "isolated" factories (that is, factories for whom all weights are 0) in the existing problem.

For instance, consider again Q_a from the previous section. The distance matrix for the follow-up test cases, $Q_a\prime$ and $Q_a\prime\prime$, $c_a\prime$, is defined as follows:

$$c_a\prime = \begin{matrix} 0 & 7 & 2 & M \\ 7 & 0 & 10 & M \\ 2 & 10 & 0 & M \\ M & M & M & 0 \end{matrix} \quad (5)$$

We can define the corresponding weight matrices, $w_a\prime$ for the zero-weight case, and $w_a\prime\prime$ for the non-zero weight case:

$$w_a\prime = \begin{matrix} 0 & 5 & 8 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}, w_a\prime\prime = \begin{matrix} 0 & 5 & 8 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{matrix} \quad (6)$$

The QAP problem $Q_a\prime$ made up of $w_a\prime$ and $c_a\prime$ will have the same total cost as Q_a. The QAP problem $Q_a\prime\prime$ defined by $w_a\prime\prime$ and $c_a\prime$ will have a total cost $M > Q_a$.

Both versions of this metamorphic relation, linking Q_a with $Q_a\prime$, and Q_a with $Q_a\prime\prime$, were implemented as described.

2.3 Merging

The previous metamorphic relations are straightforward one-to-one correspondences. As explained in section 1.2, it is possible to define metamorphic relations for the QAP involving multiple test cases. Here, we define a metamorphic relation based on the merger of two existing problem instances.

Such a merged problem Q_m can be constructed in a fairly straightforward manner from two smaller problems Q_a and Q_b. The factories for Q_m are the union of the factories from Q_a and Q_b. If $l_{a1}, l_{a2} \ldots, l_{aj}$ designate the locations in Q_a, let $l_{ma1}, l_{ma2}, \ldots, l_{maj}$ designate all the locations in Q_m taken from Q_a. For all possible pairs of $l_{ma\alpha}$ and $l_{ma\beta}$, let the distance between them be the same as the distance between the corresponding locations in Q_a, $c_{a\alpha a\beta}$. Similarly, for $l_{mb1}, l_{mb2}, \ldots l_{mbk}$, the distances between the pairs should be the same as the corresponding locations in Q_b. The weights for the merged problem should be constructed in the same manner.

The key insight enabling the optimal solution to the merged problem to be predicted is the distances and the weights between pairs, where one member of the pair is from Q_a and one from Q_b. This can be achieved by making the distance between the pairs of locations of this type some very large value M - much larger than the distances between all pairs where both are from Q_a, or both from Q_b. The weights between pairs of factories where one is from Q_a and one from Q_b are zero. To minimise the total cost, any optimal solution will ensure that the very long distances correspond with the

zero weights, and thus ensures that the factories corresponding to Q_a are in one "cluster" of locations, with the factories corresponding to Q_b at the other cluster.

If the "clusters" of factories end up at the locations corresponding to their corresponding problems, the total cost of the merged problem will be the sum of the two original problems, thus giving us a metamorphic relation. But we must guarantee that the factories are located in this manner. The simplest way to guarantee this is to ensure that the original problems are of different sizes - any attempt by the solver to place the factories in the "wrong" cluster will result in at least one very large cost penalty, and will thus not be an optimal solution.

A second metamorphic relation can be created by a technique similar to that for adding nodes, by ensuring that there is one small weight between a pair of factories, one in each cluster. One factory originally from Q_a, f_{maj}, and a factory from Q_b, f_{mbk} are randomly selected, and rather than a zero weight between them a very small weight ϵ, smaller than all existing weights, is added between them, such that $w(aj, bk) = \epsilon$. The total cost of the optimal solution to the merged problem is then the sum of the total costs of the original problems and $\epsilon \times \Delta$.

To illustrate this, we consider Q_a from the previous examples, and Q_b defined as follows:

$$w_b = \begin{matrix} 0 & 20 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 30 \\ 5 & 0 & 0 & 0 \end{matrix}, c_b = \begin{matrix} 0 & 25 & 15 & 22 \\ 25 & 0 & 12 & 44 \\ 15 & 12 & 0 & 68 \\ 22 & 44 & 68 & 0 \end{matrix} \quad (7)$$

We can now define follow-up test cases for the two variants of the metamorphic relations, Q_m for the zero-weight case and $Q_m\prime$ for the weight case. In both cases, the distance matrix c_m is defined as follows:

$$c_m = \begin{matrix} 0 & 7 & 2 & M & M & M & M \\ 7 & 0 & 10 & M & M & M & M \\ 2 & 10 & 0 & M & M & M & M \\ M & M & M & 0 & 25 & 15 & 22 \\ M & M & M & 25 & 0 & 12 & 44 \\ M & M & M & 15 & 12 & 0 & 68 \\ M & M & M & 22 & 44 & 68 & 0 \end{matrix} \quad (8)$$

The weight matrix for the zero-weight case, w_m and for the weight case, $w_m\prime$, are as follows:

$$w_m = \begin{matrix} 0 & 5 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 30 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \end{matrix} \quad (9)$$

$$w_m\prime = \begin{matrix} 0 & 5 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 30 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \end{matrix} \quad (10)$$

Q_m has a total cost equal to the sums of the costs of Q_a and Q_b, and $Q_m\prime$ has a total cost equal to $Q_m + M$.

Both of these metamorphic relations were implemented as described, with a small program written to transform two existing problem instances into a single merged one, with zero or non-zero weight between a pair of members of different original problems.

3 Experimental evaluation

As we did not have access to a set of faulty QAP solvers, we instead used the technique of mutation testing to assess the effectiveness of our metamorphic relations.

In our experiment, we compared the proportion of mutants killed by the various metamorphic relations we have devised, to the proportion killed by checking against a pre-computed correct solution - that is, with a test oracle.

3.1 Mutations - Jumble

Mutation testing is a well-established technique in which faults are deliberately inserted into software to form "mutants" to determine whether a set of test cases can detect them. It can be used for a number of different purposes, including making an assessment of the quality of test cases in terms of mutation scores, denoting the proportion of mutants killed. Jumble [9] is a mutation testing tool designed for assessing the quality of unit tests written in the JUnit unit testing framework for Java software. Jumble supports the evaluation of a test case by the following procedure:

- identifying all locations in a Java class where its supported mutation operations can be performed, producing a set of mutants.
- For each mutant, executing the specified JUnit test, and recording whether the test detects a failure.
- Reporting the proportion of mutants detected.

Jumble allows the user to specify a number of different mutation methods, but by default two types of mutations are enabled. The first involves *conditionals*, where a condition in an `if`, `while`, `do`, and `for` statements, is replaced by its negation. The second default mutation type is for arithmetic operators, where arithmetic operator is replaced by another according to a predefined table - for instance, the "+" operator is replaced with "-". Only the default mutation types were used in our experiments.

3.2 Sample cases as source test data

For our experiment, we required some QAP instances that could serve as the basis for a suitable source test set. Rather than generating our own QAP instances, we started with some sample instances from the QAPLIB library of quadratic assignment problem instances, which is widely used in the QAP research community as a testbed to determine the performance of new exact and approximate solvers. [10].

The nature of our experiment meant that even the smallest problems in QAPLIB were too slow to be practical. More tractable QAP instances were created by selecting a contiguous, randomly selected subset of the factories and locations in a problem instance in QAPLIB. The use of contiguous subsets, rather than just selecting random factories and locations, was on the basis that many of the problem instances in QAPLIB seemed to have the shortest distances to numerically contiguous factories, and many had quite sparse weight matrices, with the vast majority of non-zero weights to near neighbours. By this measure, we hoped to retain the "flavour" of the original problems.

Even so, the limited computing resources available made it impractical to apply Jumble mutations to many different test cases. Only a few test cases were therefore tried with each metamorphic relation, but using a large number of mutants.

In this study, for most problems only three problem instances were used as the source test cases (referred to as t_1, t_2 and t_3 in Table 1). 768 mutants were tried with each test case. For the "merge" relations, two source test cases are required, and that the two source test cases must be of different sizes. For these relations, three additional QAP instances, slightly smaller than t_1, t_2, and t_3, were created by the same shrinking procedure. These new QAP instances were paired with the corresponding t_i to provide the necessary second source test for the merge relation.

3.3 Testing procedure

For each selected QAP instance and each metamorphic relation (or pairs of problem instances in the case of the "merge" metamorphic relation), a JUnit test case was created. The JUnit test executes the source test case, the follow-up test case, and checks whether the specified metamorphic relation holds, and succeeds or fails accordingly. Jumble was then used to repeatedly apply the JUnit test to all the different mutated versions of the QAP solver. The proportion of mutants that were detected was recorded.

4 Results

Table 1 shows the proportion of mutants killed by running the three test cases and the various metamorphic relations. For the merge relations, as noted previously, two source test cases of different sizes were created to apply this relation. For these relations, therefore, the column t_i should be read as "t_i and another, smaller, source test case".

Metamorphic Relation	Mutants killed (%)		
	t_1	t_2	t_3
Oracle	71	73	75
Relabelling	13	13	14
Add (zero weight)	25	25	25
Add (weight)	75	74	74
Merge (zero weight)	21	22	24
Merge (weight)	77	74	76

Table 1. Proportion of mutants killed by metamorphic relations

It is not surprising that the oracle case did not kill all mutants, simply because a single test case is not necessarily a failure-revealing input for any specific mutant. What may be slightly more surprising to the reader is that the performance of two of the metamorphic relations, "Add (weight)" and "Merge (weight)" is higher than that of the oracle. This can also be explained in terms of the number of test cases executed. Effectively, "Add (weight)" executes two test cases for each source test case, and "Merge (weight)" executes three test cases for each pair of source test cases, while testing using the oracle only involves the execution of one test case. An error revealed by any of the source or follow-up test cases may violate the metamorphic relation and hence reveal the failure.

The results show dramatic differences in the effectiveness of different metamorphic relations - the "Relabelling" relation detected around 13% of mutants, whereas the "Merge (weight)" relation detected around 75% of mutants. This shows that selection of metamorphic relations is vital for the effective application of metamorphic testing

Furthermore, we observe that a seemingly minor difference in a metamorphic relation - the difference between "weight" and "zero weight" for both the "Add" and "Merge" metamorphic relations - results in dramatic differences in failure-revealing effectiveness. It is interesting to consider why this small difference contributed to such contrasting failure-revealing effectiveness. There are a number of possible explanations. One explanation, discussed in [11], is that the "weight" versions of the metamorphic relations led the program execution patterns in the follow-up test cases and the source test cases to be more divergent, giving more chance for a bug to be exposed. If so, this indicates that testers should choose relations that encourage divergent execution patterns.

There are, of course, a wide variety of other metamorphic relations that could be tried for this problem. One property of the QAP, (and many other optimization problems), is that a problem instance may have multiple optimal solutions. This makes it difficult to devise metamorphic relations based on the mapping of factories to locations, rather than the total cost. However, making metamorphic relations

conditional may make this easier. For instance, if problem instance Q has assignments a and total cost t, we create a follow-up problem instance $Q\prime$ identical to Q except that one randomly selected weight value $w_m\prime(i,j) = w_m(i,j) + \epsilon$, where ϵ is a positive constant. We can unconditionally say that $t \leq t\prime \leq t + \epsilon * c_m(i,j)$. However, two other conditional metamorphic relations exist. If the assignment for the optimal solution for $Q\prime$ is still a, the total cost c must increase by precisely $\epsilon \cdot c_m(i,j)$. Furthermore, if the cost increase is strictly smaller than $\epsilon \cdot c_m(i,j)$, $a\prime$ the assignment for the optimal solution to $Q\prime$, must be different to a.

In this experiment, we have used an exact solver that guarantees to find a globally optimal solution to the QAP. In practice, heuristic approximation algorithms are typically used; under such conditions, the metamorphic relations above would not be suitable. Some work has already been conducted in applying metamorphic testing to heuristic algorithms [12].

The dramatic difference in effectiveness of different metamorphic relations suggests it would be desirable to have some method to screen a set of metamorphic relations to find an effective subset. This study suggests that mutation analysis may prove useful for this purpose.

5 Conclusion

In this study, we have shown that metamorphic testing can be effectively used to find faults in an exact QAP solver.

To our pleasant surprise, we found that it was relatively easy, even for non-experts in the problem domain, to come up with effective metamorphic relations. Even given the relatively small scope of this study, we feel that this is a strong indication of the likely practical utility of this approach.

Our results suggest that the metamorphic testing approach is very likely to be useful for ensuring the quality of other solvers of hard optimization problems, given the difficulty of verifying the correctness of their solutions.

The simplicity of metamorphic testing suggests that relatively inexperienced practitioners, or even end users, can perform useful verification using this technique. The tester can specifically target the properties that are important for their own use of the software. Furthermore, the technique is very straightforward to describe, and requires only a small amount of domain knowledge to apply.

Acknowledgements

This project was supported by the Natural Science Foundation of China (Grant No.60421001) and Australian Research Council (ARC LX0776490).

References

[1] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," Hong Kong University of Science and Technology, Tech. Rep. HKUST-CS98-01, 1998.

[2] J. W. Gavett and N. V. Plyter, "The optimal assignment of facilities to locations by branch and bound," *Operations Research*, vol. 14, no. 2, pp. 210–232, March 1966.

[3] Wikipedia, "Quadratic assignment problem — wikipedia, the free encyclopedia," 2007, [Online; accessed 12-October-2007]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Quadratic_assignment_problem&oldid=139133382

[4] P. Pardalos, F. Rendl, and H. Wolkowicz, "The quadratic assignment problem: a survey and recent developments," in *Quadratic assignment and related problems (New Brunswick, NJ, 1993)*, P. Pardalos and H. Wolkowicz, Eds. Providence, RI: Amer. Math. Soc., 1994, pp. 1–42.

[5] H. T. Lau, *A Java Library of Graph Algorithms and Optimization*. Chapman and Hall/CRC, 2007.

[6] S. Beydeda, "Self-metamorphic-testing components," in *Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International*, Chicago, Illinois, USA, September 2006, pp. 265–272.

[7] W. K. Chan, T. Y. Chen, H. Lu, T. H. Tse, and S. S. Yau, "Integration testing of context-sensitive middleware-based applications: a metamorphic approach," *International Journal of Software Engineering and Knowledge Engineering*, vol. 6, no. 5, pp. 677–703, 2006.

[8] A. Gotlieb and B. Botella, "Automated metamorphic testing," in *Proceedings of the 27th International Computer Software and Applications Conference (COMPSAC 2003)*, 2003, pp. 34–40.

[9] Reel Two, "Jumble website." [Online]. Available: http://jumble.sourceforge.net/

[10] R. E. Burkard, S. E. Karisch, and F. Rendl, "QAPLIB - a quadratic assignment problem library," *Journal of Global Optimization*, vol. 10, no. 4, pp. 391–403, 1997.

[11] T. Y. Chen, D. Huang, T. H. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing," in *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004)*, 2004, pp. 569–583, Madrid, Spain.

[12] A. C. Barus, T. Y. Chen, D. D. Grant, F.-C. Kuo, and M. F. Lau, "Testing of heuristic methods: a case study of a greedy algorithm," in preparation.

Towards a Theoretical Model for Evaluating the Acceptance of Model-Driven Measurement Procedures

Nelly Condori-Fernández, Oscar Pastor
Department of Information Systems and Computation
Valencia University of Technology
Valencia-Spain
{nelly,opastor}@dsic.upv.es

Abstract

Software development based on transformation models has resulted in increasing interest in full automatic software measurement from conceptual models. In this paper, we propose a theoretical model for evaluation of the extent to which a model-driven measurement procedure would be accepted in practice. We identified a number of factors that could affect perceived usefulness and ease of use, and which would in turn affect the intention to use.

Keywords: *acceptance model, software measurement, model driven process, functional size*

1. Introduction

Software measurement, considered in the literature as essential for software product and process improvement, has not yet been accepted in practice. A study reported that 21.1% of software professionals contacting the Software Engineering Institute from 2004-2005 did not use a measurement method [1]. Another study showed that less than 10 percent of practitioners classified metrics programs as positive [2].

In our experience, one important reason for this gulf between researchers and practitioners is a lack of confidence among practitioners, who view manual or semi-automatic software measurement as a complex and difficult undertaking.

However, with the appearance of the model-driven development process, several proposals have arisen for automatic measurement of specific artifacts developed at early stages and in particular contexts [3][4][5][6][7].

This paper aims to explore the various factors that affect practitioners' perceptions, and how such perceptions can affect the acceptance of a model-driven measurement procedure in practice.

A number of models exist for evaluating the acceptance of new techniques and technology, in particular the Technology Acceptance Model (TAM) [8], which is considered the most applicable model in many usage scenarios. The Method Evaluation Model (MEM) [9], which uses the same TAM constructs, was the first to be applied in the context of Functional Size Measurement (FSM) procedures ([4], [10]).

From preliminary results obtained with MEM, we define an acceptance model for model driven measurement procedures, identifying particular factors that affect perceptions of usefulness and ease of use.

The structure of the paper is as follows. In section 2 we present a brief overview of the literature. Section 3 presents our proposal to evaluate the acceptance of FSM procedures. Finally, we present our conclusions and suggest further work

2. Literature Review

According to Cooper and Zmud [11], acceptance is one of the stages in the diffusion of technological innovations, and is defined from an employee perspective. This means that an organization's personnel are induced to commit to Information Technology application usage. Acceptance must not be confused with adoption; for Cooper and Zmud, adoption is defined as stage where negotiations are started in relation to the decision to adopt the innovation and mobilizing of organizational and financial resources for doing so.

The acceptance of technology has been investigated in a number of different fields (Internet-based health applications [11], multimedia information systems for learning [12], CASE tools [13], etc.). However, in the software measurement field few papers on this subject were found in the literature.

* This work has been Supported by the SESAMO project, ref. TIN2007-62894 and co-financed by FEDER.

Among the relevant literature is the proposal of Umarji and Emurian [14] focused on evaluation of the likelihood of acceptance of a metrics program. Their model takes as input organizational culture, and the nature of the metrics program.

Gopal et al. [15] researched the influence of institutional factors on the assimilation of metrics in software organizations. They also identified a set of determinants for metrics program success [16]. These determinants are divided into organizational and technical variables.

In our research we do not deal with in-depth organizational variables, since our proposal focuses on measurement procedure acceptance from a software practitioner's perspective and on the intrinsic nature of software measures.

To define our acceptance model, we first present our experience using MEM [9] in the functional size measurement context.

3. A Theoretical Model for Evaluating the Acceptance of FSM Procedures

In this section, we define a model for evaluating the acceptance of FSM procedures in practice by means of the extension of a theoretical model.

3.1. Previous research: MEM and FSM

In the last seven years we have been working on functional size measurement at early stages of the model-driven development process, defining and empirically evaluating three FSM procedures: 1) OOmFP [4], a measurement procedure designed for sizing conceptual schemas in function points; 2) OOmFPweb [4], extended procedure to size Web applications; and 3) RmFFP [6], a procedure designed to measure size from object-oriented functional requirements specifications.

The empirical evaluation of these FSM procedures was carried out by applying MEM [9], with the results of this evaluation reported in [4] and [10]. MEM combines Rescher's theory of pragmatic justification [17], and Davis's Technology Acceptance Model (TAM) [8]. The core of the MEM consists of the same perception-based constructs as the TAM, but which were adapted to evaluate Information System design methods. These constructs are called the Method Adoption Model (MAM).

- *Perceived Ease of Use*: the extent to which a person believes that using a particular method would be free of effort.
- *Perceived Usefulness:* the extent to which a person believes that a particular method will be effective in achieving intended objectives.
- *Intention to Use:* the extent to which a person intends to use a particular method.

We used MEM in its original form, and analyzed its applicability to functional size measurement, based on an integrative review of these three empirical evaluations reported in [18].

Only one of the MEM relationships was corroborated. This means that the intention to use an FSM procedure is influenced more significantly by perceived usefulness than by perceived ease of use. In addition, the relationship between perceived usefulness and ease of use was not significant for the FSM procedures context.

We therefore came to the conclusion that perceived usefulness should not only be evaluated with respect to the effectiveness of the FSM procedure in achieving objectives intended by the users, but that other factors should be included.

3.2. Extending the MAM to evaluate FSM procedures

We have extended the MEM core (MAM) by the inclusion of the factors that influence perceptions of usefulness and ease of use when users are using a model driven FSM procedure (see Figure 1). Two types of factors were identified:

- *Intrinsic Factors* corresponding to quality and tangibility of results, and the minimum number of actions required for calculating the functional size using an automated measurement procedure.
- *Extrinsic Factors* corresponding to the experience and job relevance of the software practitioner.

Next, we describe each of the factors included in our proposed model.

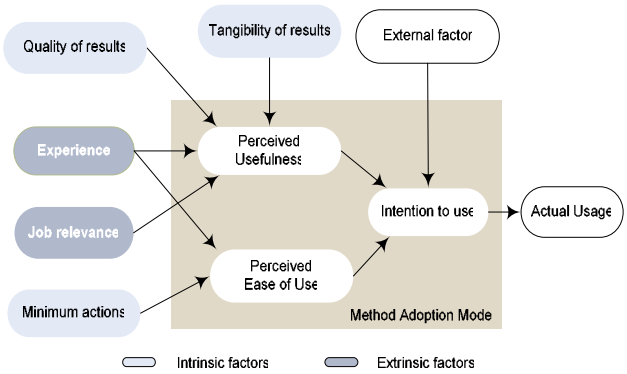


Figure 1. Model for evaluating acceptance of an FSM Procedure

3.2.1. Quality of results. According to the ISO/IEC 14143-3 report [19], certain performance properties of an FSM method have to be taken into account when analyzing the quality of results. These properties are

expressed in terms of accuracy, repeatability, reproducibility and convertibility:

Accuracy: closeness of agreement between a quantity value obtained by measurement and the true value of the measurand.

Precision: closeness of agreement between quantity values obtained by repeated measurements of a quantity, under specified conditions, i.e. repeatability and reproducibility.

Convertibility: this is defined as the ability to convert the results obtained by applying two or more FSM Methods in the measurement of the same set of Functional User Requirements.

Automated measurement achieves precision. Therefore, we consider accuracy and convertibility as quality properties that may influence the usefulness perceived by the users when they use a Model-Driven FSM procedure. We therefore propose that:

P1: There will be a positive relationship between Quality of results and Perceived Usefulness

3.2.2 Tangibility of results. This factor is also called result demonstrability, which has been refined by Moore and Benbasat for information systems and is derived from Innovation Diffusion Theory used in the sociology field [20].

In our case, even if a measurement procedure produces effective (accurate, precise and convertible) results, if these results are not interpreted by an indicator (they are "indistinct"), users will have difficulty understanding the usefulness of the measurement procedures.

For this reason, we have redefined this factor as the extent to which an individual believes that the results of using a FSM procedure can be observable and understandable. Thus we propose:

P2: There will be a positive relationship between Tangibility of Results and Perceived Usefulness

3.2.3. Minimum actions. Previous empirical studies, [4], [23] corroborated that for manual measurement, *measurement productivity*[1] has a positive effect on perceived ease of use. However, this variable is irrelevant for fully automated FSM procedures. We believe that perceived ease of use could be influenced by the minimum number of actions necessary for the obtaining of the functional size. Therefore, we propose:

P3: There will be a positive direct relationship between Minimum Number of Actions required to obtain the functional size and perceived ease of use.

3.2.4 Job relevance. It is possible for a model-driven measurement procedure not to be perceived as useful even though the procedure provides accurate and convertible results, possibly because the use of the FSM procedure is not relevant for the job type (e.g. executive, project manager, designer, etc.) of the software practitioner concerned.

We have redefined job relevance as the extent to which an individual believes that an FSM procedure is applicable and relevant to his or her job. Thus we intend to examine whether:

P4: There will be a positive relationship between Job Relevance and Perceived Usefulness

3.2.5 Experience. Various studies assert that subjects with direct hands-on experience would be more likely to hold stronger perceptions as to ease of use and perceived usefulness of a technology, based on the subjects' ability to generate more beliefs and to extrapolate from past behaviors related to their experience [21],[22]. We hypothesize that measurement and modeling experience should have a strong effect on user's perceptions. For instance, less experienced modelers may be unable to adequately understand the software artifact to be measured, which could cause erroneous perceptions regarding usefulness and ease of use of an FSM procedure.

We define the experience factor as knowledge or skill gained in use measurement and development methods over a period of time.

Thus we propose the following:

P5: There will be a positive relationship between measurement and modeling Experience and Perceived Ease of use.

P6: There will be a positive relationship between measurement and modeling Experience and Perceived Usefulness.

3.2.6 External factors. These are factors that do not depend on the measurement procedure in itself, but on the organization as a whole. These include where the business follows trends in the market based on advertising and marketing or peer company use, or has business priorities giving rise to time or cost constraints, or the maturity level of an organization [16]. We intend to determine whether these external factors influence the intention to use a Model Driven FSM procedure. Thus, we propose the following premise:

P7: There will be a relationship between External Factors and the intention to use.

4. Conclusions and further work

We have defined a theoretical model to evaluate the acceptance of model-driven measurement procedures from an individual perspective. This model was

[1] Number of size units that can be measured per unit of time.

defined by means of the extension of the MEM core (MAM), by including two types of factors that influence perceptions of usefulness and ease of use (intrinsic and extrinsic factors).

We have considered result quality to be a primary and intrinsic factor that will affect the intention to use model driven measurement procedure. As these results cannot in themselves be interpreted, another intrinsic factor was used to evaluate whether obtained results are understandable by practitioners when using an estimation model (are tangible). Finally, as this measurement is automated, manual measurement productivity has not been considered. The minimum actions required to obtain functional size in a model driven context have been taken into account.

With respect to extrinsic factors that will influence perceptions of usefulness and ease of use, we have considered the factors that affect the practitioner's viewpoint when using a measurement procedure (e.g. level of experience, job relevance and organizational factors).

Finally, we plan to carry out an empirical study to verify causality relationships between both extrinsic and intrinsic factors and .the MAM constructs.

References

[1] Kasunic M., State of Software Measurement Practice Survey, Carnegie Mellon, Software Engineering Institute, 2006, www.sei.cmu.edu/sema/presentations/stateof-survey.pdf

[2] Daskalantonakis M., A Practical View of Software Measurement and Implementation Experiences Within Motorola, IEEE Trans. Software Eng., vol. 18, pp.998-1010, Nov. 1992.

[3] Abrahão S., Gomez J., Insfran E. Mendes E., A Model-Driven Measurement Procedure for Sizing Web Applications, Conference on Model-Driven Engineering Languages and Systems (MODELS 2007), Nashville, TN, USA, September 30-Octuber 5, 2007, LNCS Springer, 2007.

[4] Abrahao S., Poels G., Pastor O. A Functional Size Measurement Method for Object-Oriented Conceptual Schemas: Design and Evaluation Issues. Software & System Modelling, 5(1): 48-71, Springer Verlag, 2005.

[5] Azzouz S., Abran A., "A Proposed Measurement Role in the Rational Unified Process and its Implementation with ISO 19761: COSMIC-FFP" in Software Measurement European Forum, Rome, Italy, 2004.

[6] Condori-Fernández N., Abrahão S., and Pastor O., On the Estimation of Software Functional Size from Requirements Specifications, Journal of Computer Science and Technology (JCST), Springer, 22(3): 358-370, 2007.

[7] Abrahão S., Condori N., Olsina L., and Pastor O., Defining and Validating Metrics for Navigational Models, 9th IEEE International Software Metrics Symposium September 2003, Sydney, Australia, IEEE Press, pp. 200-210.

[8] Davis F. D., "Perceived Usefulness, Perceived Ease of Use and User Acceptance of Information Technology", MIS Quarterly, vol. 3, no. 3, 1989.

[9] Moody D. L., The method evaluation model: a theoretical model for validating information systems design methods, 11th European Conference on Information Systems, ECIS 2003, Naples, Italy 16-21 June 2003.

[10] Condori-Fernández N., Pastor O., An Empirical Study on the Likelihood of Adoption in Practice of a Size Measurement Procedure for Requirements Specification, Sixth International Conference on Quality Software (QSIC 2006), October 2006, Beijing, China, pp. 133-140.

[11] R.B. Cooper and R.W Zmud, "Information Technology Implementation Research: A Technological Difussion Approach", Management Science, 36(2):123-139, 1990.

[12] W. G. Chismar, S. Wiley-Patton, Does the Extended Technology Acceptance Model Apply to Physicians, 36th Annual Hawaii International Conference on System Sciences, IEEE Computer Society, Big Island, USA, January 2003, pp. 160-167.

[13] Chau P.Y. K., An empirical investigation on factors affecting the acceptance of CASE by systems developers, Journal on Information and Management, Elsevier, 30(6): 269-280, 1996.

[14] Umarji M.and Emurian H., Acceptance Issues in Metrics Program Implementation, Proceedings of the 11th IEEE International Software Metrics Symposium, IEEE Computer Society, 2005, Washinton,USA, pp. 10-29.

[15] Gopal A., Krishnan M.S., Mukhopadhyay T., Impact of Institutional Forces on Software Metrics Programs, IEEE Trans. on Software Eng, 31(8):679-695, August 2005.

[16] Gopal A., Krishnan M.S., Mukhopadhyay T., and Goldenson, Measurement Programs in Software Development: Determinants of Succes, IEEE Transaction on Software Eng., 28(9):863-875, 2002.

[17] Rescher, N., Methodological Pragmatism: Systems-Theoretic Approach to the Theory of Knowledge, Oxford: Basili Blackwell, 1977.

[18] Condori-Fernández N., Pastor O., Analyzing the Applicability of a Theoretical Model in the Evaluation of Functional Size Measurement Procedures, Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering, SEKE 2007, Boston, Massachusetts, USA, July 9-11, 2007, pp. 736-739.

[19] ISO, "ISO/IEC 14143-3: Information technology - Functional size measurement - Part 3: Verification of functional size measurement methods", 2003.

[20] Rogers, E. Diffusion of Innovation. 4th ed. Free Press, New York, 1995.

[21] Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. User acceptance of information technology: Toward a unified view. MIS Quarterly, (27:3), 425-478, 2003

[22] Irani T., If we build it, will they come? The effects of experience and attitude on traditional-aged students' views of distance education, International Journal of Educational Technology, 2(1):1-12, July 2000.

[23] Condori-Fernández N., Pastor O., Evaluating the Productivity and Reproducibility of a Measurement Procedure. ER Workshop on Quality on Information System, LNCS Springer, 2006, Tucson Arizona, pp. 352-361.

Knowledge Transformation from Task Scenarios to View-based Design Diagrams

Nima Dezhkam and Kamran Sartipi
Dept. Computing and Software, McMaster University, Hamilton, ON. L8S 4K1, Canada
{*dezhkan, sartipi*}*@mcmaster.ca*

Abstract

A large body of research in software requirement engineering domain has been dedicated to enhancing the structure of task scenarios using scenario schemas and predefined structures. However, less attention has been paid to the application of schemas in extracting design knowledge from scenarios. In this paper, we propose a schema-based technique to extract the design knowledge embodied in the text of scenarios and represent them using multi-view design diagrams. In this context, we define a framework and a scenario syntax that allow for generating a set of structured scenarios that cover the requirements of a software system. We define a novel scenario schema to parse the informal text of scenarios and populate an objectbase to maintain the design knowledge building blocks. Consequently, a set of guidelines are defined to incrementally build design diagrams for software views such as data and function. As a case study, the design diagram generation for a restaurant system is presented.

KEYWORDS: Knowledge; Transformation; Scenario; Schema; Design; Multiple Views; Object base.

1 Introduction

Scenario-based knowledge extraction from requirements has attracted significant attention within the requirement engineering field [13]. Scenarios are represented in a variety of formal and informal methods ranging from simple text and graphical media to relational algebra [6]. In this paper, we define a scenario as "*a structured narrative text describing a system's requirements in terms of system-environment interactions at business rule level*". Scenarios are considered as easy-to-use and effective means in different phases of software engineering process, such as: requirement elicitation and analysis, design representation, code development, testing, and maintenance [11, 8, 10, 14]. A wide range of research in knowledge extraction from software requirements attempt to investigate: the enhancement of scenario generation by using scenario schemas or pre-defined structures [3, 17]; scenario analysis and knowledge extraction [2]; and design-related document generation [15, 16].

In this paper, we introduce a novel technique to transform the knowledge from scenarios into well-formed design diagrams in two views of data and function. In this technique scenarios are generated using domain knowledge and in conformance with a regular expression syntax that imposes a structure to the scenario representation. The proposed approach allows us to reuse the domain knowledge and business rules within the scenarios through a scenario template knowledge base. Further, the generated structured scenarios are parsed using a novel *scenario schema* to populate an objectbase of design related entities and dependencies. The populated objectbase serves both as a data source during the design diagram construction and as a valuable electronic asset of design knowledge to be analyzed, augmented, and used during the maintenance phase of the software system. Finally, the information in the objectbase is used to create standard diagrams, such as Entity-Relationship diagram (ER) for data view, and function diagram for function view.

The contributions of this paper include: i) a framework to transform the structured knowledge of the scenarios into view-based design diagrams; and ii) a novel schema that allows for decomposing the scenarios into an objectbase of design-related entities and dependencies. As a case study, the design diagram generation for a fast-food restaurant system is presented.

2 Related work

The proposed approach in this paper relates to the literature for capturing and representing knowledge from task scenarios for various purposes. We present several approaches and discuss their similarities and contrasts with our work.

Anton and Potts [1] discuss different representations of scenarios in object oriented software engineering and requirements engineering. Jarke et al. [9] present a review on approaches to scenario-based requirement engineering and research issues.

Lamsweerde et al. [5] introduces KAOS methodology that supports requirements extraction from high-level goals, and assigns objects and operations to the various agents in a system. Their meta-model has similarities with our schema, however our approach aimed at extracting design diagrams after capturing the requirements.

In [6] a formal representation of scenarios using tabular expression is introduced in order to simplify the tasks of scenario validation, verification, and integration. In [3] a schema for semantic model of scenarios is defined to help requirement refinements. Leite et al. [7] aid the process of scenario construction and management by structuring scenarios using a conceptual model along with a form-oriented language. However, in addition to requirement elicitation and validation, our framework transforms the generated structured scenarios into design diagrams. Damas et al. [4] propose tool-supported techniques to generate behavior models from end-user scenarios, whereas we extract design diagrams from scenarios. Hufnagel et al. [16] present a scenario-driven object oriented requirements analysis to support design of a system. This approach does not define a scenario schema and also it is methodology dependent. In [12] a method for modular representation of the scenarios is proposed that supports the reusability of the scenarios in different design contexts. This approach is similar to ours in the sense that it attempts to define a structure for the scenarios.

Overall, the significance of our approach is that we generate scenarios using semi-structured templates and transform the knowledge within the text of scenarios into design relevant knowledge using guidelines that provide a repeatable and view-based design reconstruction process.

3 Proposed framework

In this section, we discuss the steps for transformation of the knowledge embodied in the text of scenarios into design knowledge represented by two views data and function of a software system. These steps are presented using the framework of Figure 1. In a nutshell, the proposed framework generates a set of structured scenarios and uses a schema to parse these scenarios into ingredients of the view-based design representations. The proposed framework consists of three stages, as follows.

3.1 Stage 1: scenario generation

This stage consists of generating a set of structured text-based scenarios that conform with a regular expression syntax. To facilitate scenario generation and controlling the format and vocabulary of the generated scenarios, a predefined set of domain-specific templates can be utilized.

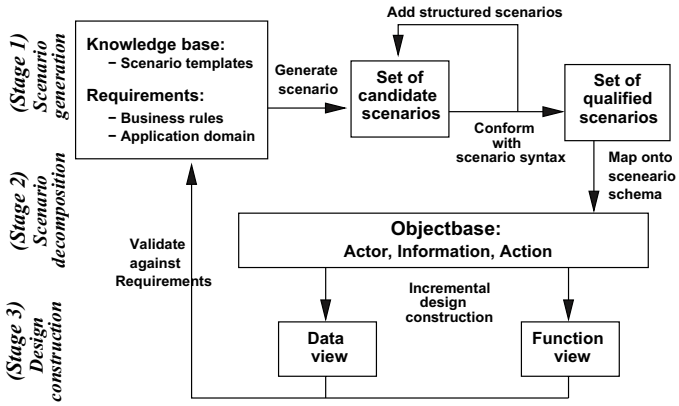


Figure 1. The proposed design construction framework from scenarios.

Consequently, at the end of this stage a set of qualified scenarios are produced that cover a part or the whole of the system requirements.

Scenario structure. We define a structure for scenarios that is imposed by the regular expression syntax in Figure 2 and the semantics that are defined by the application domain's business rules. In this scenario syntax, *Actor, Action*, and *WorkingInformation* are the entity-types and action-types that will be defined in Section 3.2. Each scenario consists of a sequence of one or more *Actors*, *Actions*, and *Working Information*, each of which can have between zero or more *Constraints*. In this form we can generate syntactically correct scenarios which will be further decomposed to populate the objectbase in Section 3.2 and generate design diagrams in Section 3.3.

Scenario templates. In order to facilitate generation of structured scenarios and reuse of the captured domain knowledge and vocabulary, the proposed framework leverages a tool to populate a knowledge-base of scenario templates which are organized to store the structured scenarios in a specific application domain. This allows a software engineer to assemble scenarios using a repository of domain-specific vocabulary that is maintained for a software domain. Figure 3 illustrates a sample scenario template form for a fast-food restaurant system. This form consists of fields such as: Actor, Information, and Action, where each field possesses a vocabulary of corresponding business terms. The generated scenario at the bottom of the form is a proper assembly of the terms selected from these fields.

$$Scenario : \{Actor + \{Constraint\}^{0..N}\}^{1..N} + \{Action + \{Constraint\}^{0..N}\}^{1..N} + \{Working\ information + \{Constraint\}^{0..N}\}^{1..N}$$

Figure 2. Regular expression syntax for scenario generation, where "+" and "0..N" represent composition and range, respectively.

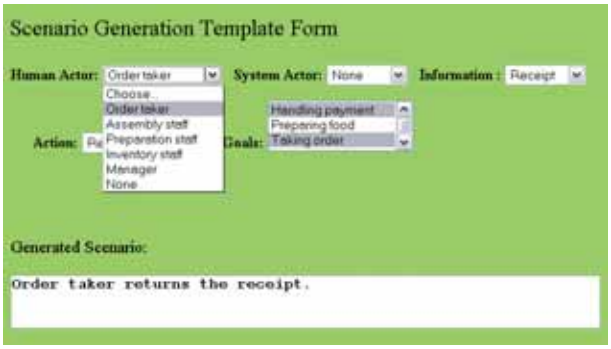


Figure 3. Scenario generation template form for a fast-food restaurant system.

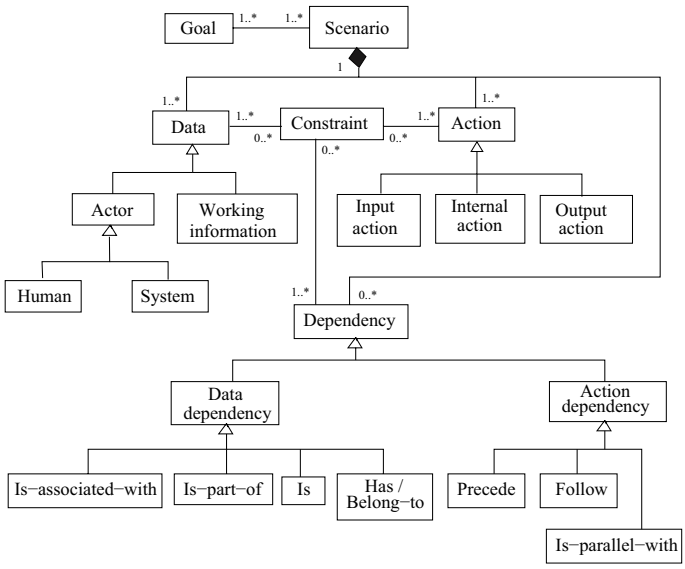


Figure 4. Scenario Schema to parse a scenario and populate an objectbase.

3.2 Stage 2: scenario decomposition

In this stage, the qualified scenarios are mapped onto the proposed scenario schema in Figure 4 which allows us to parse the structured scenarios and generate instances of classes Goal, Actor, Working information, Action, and their corresponding dependencies that are defined in the scenario schema. The generated instances incrementally populate an objectbase of design knowledge that is used to generate design-related diagrammatic representations.

Using the class diagram representation of the scenario schema in Figure 4 the texts of the structured scenarios are parsed and the resulting instances of the classes are stored in the objectbase. The objectbase is represented as a group of columns, where each column stores the instances of a class in the scenario schema that belong to different scenarios. In other words, a scenario (as a row) in the populated objectbase consists of the instances of the relevant classes of the scenario schema that are stored in different columns, and a unique *index* that identifies the scenario.

As shown is Figure 4, in our model every instance of the *Scenario* class is composed of one or more instances of *Actor*, *Working information*, and *Action* classes, and zero or more instances of *Dependency* class. Every Action, Actor, and Working information is associated with zero ore more *Constraint*s. Moreover, every *Scenario* instance is associated with one or more instances of *Goal* class. In the rest of this section the classes of the proposed scenario schema are introduced along with examples from a fast-food restaurant system domain.

Goal: represents the reasons and the desired effects for which the subject system has been produced and used. A goal can be *functional* which corresponds to performing a task, or *objective* which refers to achievement of a quality for the system. Examples of goals in a fast-food restaurant system are as follows: handling payment (functional), preparing food (functional), and shortening order preparation time (objective).

Actor: an actor is a "human" or a "system" or a "component of a system" that interacts with other actors during the execution of the scenarios. Examples of actors in a restaurant system include: order taker (human), raw material supplier (system), or food assembly station (component of a system).

Action: an action is an activity that is performed by an actor during the execution of the scenarios. Generally, an

action manipulates an instance of *Working information*. Actions can be categorized into three different types of *Input, Internal*, and *Output*, based on the scope of the system. Examples include: taking order (input), computing the price of an order (internal), and delivering food (output).

Working information: refers to the information that is manipulated (exchanged, transported, communicated, operated on, stored, etc.) by the scenario's actor during the execution of the scenario's actions. Examples are: customer's order, raw material, menu item, and item price.

Dependency: refers to a binary relationship between two instances of the classes *Actor, Action*, or *Working information*. When needed, the multiplicity of the participants in a dependency should be mentioned in the dependency instance. In such a case, the dependency can be represented by a quadruple with the multiplicity of each participant proceeding it.

In our schema, a dependency can be of type *Data dependency* or *Action dependency*. Data dependency can be one of the following subtypes: *Is*, e.g., "order taker *Is* an employee"; *Is-associated-with*, e.g., "every menu item *Is-associated-with* a recipe" (or (1,menu item,1, recipe)); *Has*, e.g., "every menu item *Has* a name"; *Belong-to*, that is the inverse[1] of *Has*, e.g., "an ID *Belongs-to* an employee"; *Is-part-of*, e.g., "a kitchen *Is-part-of* a restaurant".

Action dependency can be one of the following subtypes: *Precede*, e.g., "order payment *Precedes* order delivery"; *Follow*, that is the inverse of *Precede*, e.g., "order preparation *Follows* order taking". *Is-parallel-with*, e.g., "sending order to assembly station *Is-parallel-with* sending order to preparation station".

The proposed scenario schema in Figure 4 includes a *Constraint* class that associates any quantifiable constraint to *Data*, *Action*, and *Dependency* classes. Examples of different types of constraints include: *capacity*, *value range*, *ordinal*, *timing*, *privilege*, etc. As an example, a restaurant system may have "*younger than 10*" as a *constraint* associated with *actor* of some scenario, in order to perform a specific *action* such as "offering kids deal".

3.3 Stage 3: design construction

In this section, we discuss the guidelines that transform the contents of the objectbase obtained in Stage 2 into design diagrams. Entity-Relationship (ER) and function diagrams are the most intuitive and relevant diagrams that can be directly extracted from the objects within the objectbase and represent data view and function view of the system, respectively.

Data view. The following guidelines specify the generation of ER diagrams from the objectbase:

Data view step I. Extract all instances of *Actor*, *Working information*, and *Data dependency* classes from the object base and apply the following rules on them:

1. Instances of *Actor* and *Working information* are candidate entities/attributes.
2. Instances of *Is* dependency imply generalization and inheritance relationships, i.e., A *Is* B, means A is sub-entity of B, or B is super-entity of A.
3. Instances of *Is-associated-with* dependency imply candidate association relationships.
4. Instances of *Has* and *Belong-to* dependencies are used to identify the attributes of the entities, i.e., A *Has* B (or B *Belongs-to* A) means B is an attribute of entity A.
5. Instances of *Is-part-of* dependency imply candidate decomposition relationships.
6. Candidate entities/attributes that never appear on the right-hand side of a *Has* dependency (or left-hand side of a *Belong to*) dependency are entities and not attributes.
7. Candidate entities/attributes that appear on either side of a *Is*, *Is-associated-with*, or *Is-part-of* relationship are considered as entities.
8. Candidate entities/attributes that appear on the left-hand side of a *Has* dependency (right-hand side of a *Belong to* dependency) are considered as entities.
9. Candidate entities/attributes that appear on the right-hand side of a *Has* dependency (or left-hand side of a *Belong to* dependency) and do not apply in any of the rules vi-viii, are considered as the attributes of the entity on the other side of that dependency.

Data view step II. Depict every entity by a rectangle, every attribute of an entity as a bubble connected to it and label them by their names. Every relationship between two entities can be represented by a line connecting them. Label every relationship according to the type of dependency it came from, e.g., "is", "is-part-of", etc.

Function view. Function view of a system is well represented by function diagrams. The following guideline specifies the generation of function diagrams from the objectbase.

Function view step I. Extract all instances of *Action*, *Action dependency*, and *Constraint* classes from the object base and apply the following rules on them:

1. Instances of *Action* class are the functions.
2. Instances of the *Follow* and *Precede* dependencies determine the time-order of execution of the functions. To simplify the diagram generation, transform all the *Precede* dependencies to *Follow*, i.e., for all functions f_1 and f_2, change $f_1 Precede f_2$ to $f_2 Follow f_1$.

[1] For some dependencies, their inverse dependencies are also included in the schema to facilitate back tracing of dependencies in the objectbase.

3. The participants of a *Is-parallel-with* dependency must be executed concurrently.
4. The condition(s) for a function to follow another is determined by the *Constraint*s related to the function, actor, and working information in the corresponding scenario that the "following" appears.

Function view step II. Generate $Follow^+$ relationship (the transitive-closure of the *Follow*), i.e., $f_1 Follow^+ f_2$ means there exists a set of functions g_i where, f_1 *Follow* g_1, g_1 *Follow* g_2, ..., g_n *Follow* f_2.

Function view step III. Sort the functions in ascending order based on the number of the functions they follow, i.e., based on the number of times they appear on the left hand side of a *Follow* relationship.

Function view step IV. Start from the beginning of the sorted list, depict the first function (name A) with a square and label it by its name. List all the functions that *Follow* A. If the list contains only one function (name B), depict B and connect A to B with an arrow. If the followers list contains more than one function (name B, C, ...), then a choice condition has occurred. If there are any pair of functions (name B and C) in the list that have an *Is-parallel-with* dependency, connect A to B and C with arrows and an *AND* bubble. Otherwise the functions are connected using an *OR* bubble. Next, all arrows are labeled with the triggering condition(s) obtained in rule "4" above. Finally, remove A from the list and repeat *Function view step IV*, until the list is empty.

The functions in the function view correspond to the actions performed by the actors in the system, and can be considered as candidate methods of classes in the detailed design of the system. Also, the sequence and the AND and OR relationships between the functions reflect the design decisions that should be considered during the implementation phase of the system.

The above guideline can be semi-automated. User involvement is required in cases of conflicts or inconsistencies, such as duplicate usage of actor or action names in different roles, etc. In such cases user can be prompted to perform manual resolution.

The realization of the scenario to design transformation will be presented as a case study in the next section.

4 Case study: Fast-food Restaurant System

In this section, the results of applying the proposed framework to the case of a fast-food restaurant system is presented.

4.1 Stage 1: scenario generation

The following scenarios that conform with the proposed scenario syntax in Figure 2 were generated using our proprietary scenario generation tool. Note that for simplicity in demonstration, the following scenarios demonstrate little interactions and few conditions.

• Scenario #1: *"Order taking station computes and reports the price of the orders."*
• Scenario #2: *"Order taking station sends the paid orders to assembly station."*
• Scenario #3: *"Order taker logs into the OT station using ID and password."*
• Scenario #4: *"Order taker initiates orders."*
• Scenario #5: *"Order taker adds and removes (edit) menu items of an unpaid order."*
• Scenario #6: *"Order taker enters the amount of money received from the customer (cash-in) to OT station."*
• Scenario #7: *"Order taker defers the payment of orders."*
• Scenario #8: *"Order taker reviews the orders."*
• Scenario #9: *"Order taker calls-back unpaid orders."*
• Scenario #10: *"Order taker returns the change (and receipt) for the order."*
• Scenario #11: *"Order taker sends the cash exceeding cash limit to the cash safe."*
• Scenario #12: *"Order taker logs out from his/her ID."*

4.2 Stage 2: scenario decomposition

At this stage, the scenarios were mapped onto the proposed scenario schema to instantiate different class instances and the resulting instances are stored in the objectbase. Table 1 presents a part of the objectbase that is populated with instances of *Data* and *Action* and five *Dependency* classes from Scenarios #1 to #10 above.

4.3 Stage 3: design construction

In this stage we followed the guideline presented in Section 3.3 to construct the diagrams for data and function views.

Data view. Candidate entities/attributes are stored in different *Data* columns (i.e., $Actor|_{System}$, $Actor|_{Human}$, and $Working\ information$) of the objectbase. Similarly, the dependencies among these candidates are stored in the objectbase (under *Is*, *Belong-to*, ... columns). A part of the the ER diagram for the restaurant system (i.e., order taker component), constructed using the guideline for **Data view** is shown in Figure 5.

Function view. The list of extracted functions sorted by $Follows^+$ relationship is shown in Table 2. Also, the extracted dependencies between these actions are stored in different *Action dependency* columns of the objectbase. The function diagram for the order-taker component of the restaurant system (constructed using the guideline for **Function view**) is shown in Figure 6.

Table 1. A part of the objectbase created from the scenarios #1 to #10.

$Index$	$Actor\|_{System}$	$Actor\|_{Human}$	$Working\ information$	$Action\|_{Input}$	$Action\|_{Internal}$	$Action\|_{Output}$
1	OT Station	-	order,price	-	compute price	report price
2	OT Station, ASM station	-	paid order	-	-	send paid order to ASM station
3	-	order taker,OT station	ID&password	-	login to system	-
4	-	order taker	order	-	initiate order	-
5	-	order taker	menu item,unpaid order	-	add/remove menu item	-
6	-	order taker,OT station	cash-in	enter cash-in	-	-
7	-	order taker	order	-	defer payment	-
8	-	order taker	order	-	review	-
9	-	order taker	unpaid orders	-	call-back	-
10	-	order taker	change/receipt	-	-	return change/receipt

Index	Is-associated-with	Belong-to	Is-part-of	Follow	Precede
1	-	(price,order)	(report price, compute price)	(report price, compute price)	-
2	-	-	(1,paid order,1,order)	(send paid order to ASM station, report price), ...	-
3	-	(ID&password,order taker)	-	-	(login to system, send paid order to ASM station), ...
4	(1,order taker,n,customer order)	-	-	(initiate order, login to system)	(initiate order, compute price)
5	(n,menu item,1,order)	-	-	(edit order, initiate order), ...	(edit Order, compute price), ...
6	-	(cash-in,order)	-	(enter cash-in, report price), ...	(enter cash-in, send paid order to ASM station), ...
7	-	-	-	(defer payment, edit order), ...	-
8	-	-	-	(review orders, login to system)	-
9	-	-	(1,unpaid order,1,order)	(call-back unpaid orders, login to system)	(call-back unpaid orders, enter cash-in), ...
10	-	(change/receipt,order)	-	(return change/receipt, enter cash-in), ...	(return change/receipt, send paid order to ASM station)

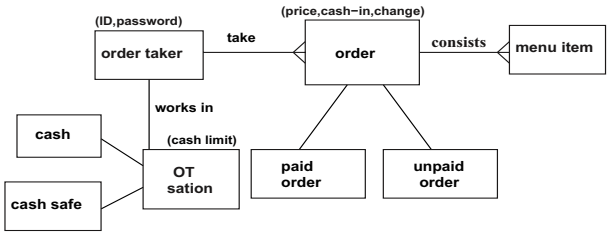


Figure 5. Generated Entity-Relationship diagram for the order taking component.

The generated design diagrams and the existing knowledge in objectbase will enable us to extract other design diagrams such as class diagram of the system. Figure 7 illustrates the complete class diagram of the restaurant system. This diagram is obtained from the ER diagram of the system that was generated in the proposed framework. The space limitation of the paper does not allow us to provide the required guidelines.

5 Discussion and conclusion

In this paper, we presented a systematic and semi-automatic approach for transforming the design knowledge within task scenarios onto a set of design diagrams. We proposed a framework with three major stages of scenario generation, scenario decomposition, and design construction.

Table 2. List of actions in order taking component and corresponding to *Follow* relation.

Index	Action	$Follows^+$
1	Login using ID & password	-
2	Logout the system	1
3	Review orders	1
4	Initiate order	1
5	Call-back unpaid orders	1
6	Edit orders	1,5
7	Compute price	1,5,6
8	Report price	1,5,6,7
9	Defer order payment	1,5,6,7,8
10	Enter cash-in	1,4,5,6,7,8
11	Return change & receipt	1,4,5,6,7,8,10
12	Send order to assembly station	1,5,6,7,8,10,11
13	Send excess cash to cash safe	1,4,5,6,7,8,10,11,12

The task scenarios are structured by the means of a regular expression syntax and can be reused through a knowledge-base of scenario templates. A scenario schema has been proposed as the core of the approach that allows us to decompose scenarios into design entities and dependencies as the means to populate an objectbase. The generated objectbase would maintain the building blocks that allow the

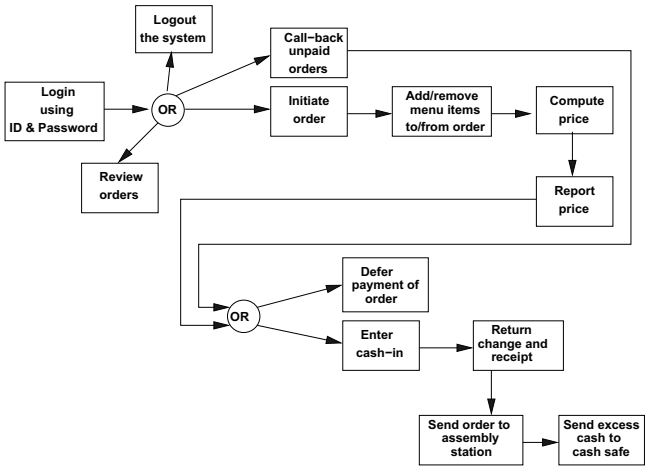


Figure 6. Generated function diagram for order taking component.

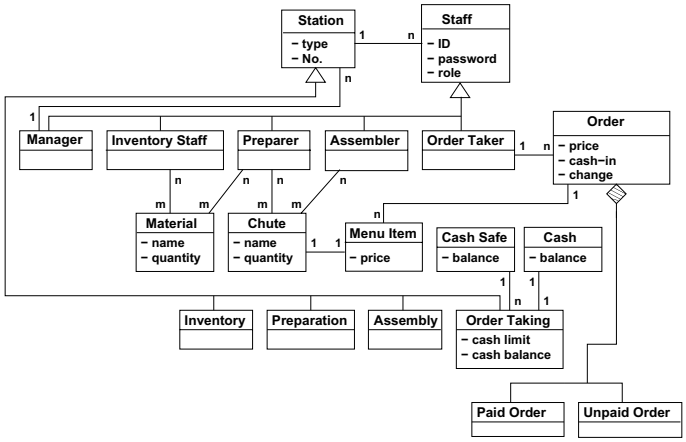


Figure 7. Generated class diagram of the whole restaurant system.

engineer to generate the design diagrams for two views of the software system using common-practice modeling.

We compared the constructed Entity Relationship diagram in Figure 5 with the similar diagram generated for the same restaurant system by a software engineer. In this comparison 6 out of 8 entities for the order taking component were the same in both diagrams which indicates a promising result. The proposed technique provides a disciplined and structured approach to requirement-to-design transformation process within the knowledge engineering field. The proposed scenario schema provides a clear understanding of the major building blocks of the software system's functional entities and their relationships. The populated objectbase serves both as a data source during the design diagram construction and as a valuable electronic asset of design knowledge to be analyzed, augmented, and used during the maintenance phase of the software system. Specifically, the objectbase can be mined to extract more general design decisions that is not feasible by a human-based analysis.

References

[1] A. Anton and C. Potts. Representational framework for scenarios of system use. In *Requirements Engineering*, volume 3, pages 219–241, 1998.

[2] L. Chung and K. Cooper. A knowledge-based cots-aware requirements engineering approach. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 175–182, New York, NY, USA, 2002. ACM Press.

[3] C.Potts. Scenic: A strategy for inquiry-driven requirements determination. In *Proc. RE'99: International Symposium on Requirements Engineering*, Limerick, Ireland, June, 1999.

[4] C. Damas, B. Lambeau, and P. Dupont. Generating annotated behavior models from end-user scenarios. *IEEE Trans. Softw. Eng.*, 31(12):1056–1073, 2005.

[5] R. Darimont, P. Massonet, and A. Van Lamsweerde. KAOS: An Environment for Goal-Driven Requirements Engineering. *In Proceedings of the ICSE'98*, pages 1–2, 1998.

[6] J. Desharnais, R. Khedri, and A. Mili. Representation, validation and integration of scenarios using tabular expressions. *Journal of Formal Methods in Software Development. Special issue on tabular expressions*, 2002.

[7] J. C. S. do Prado Leite, G. D. S. Hadad, J. H. Doorn, and G. N. Kaplan. A scenario construction process. *Requirements Engineering*, 5(1):38–61, 2000.

[8] Haumer, P. Pohl, and K. Weidenhaupt. Requirements elicitation and validation with real world scenes. In *IEEE Transactions on Software Engineering 24*, pages 1036–1054, 1998.

[9] M. Jarke, T. X. Bui, and J. M. Carroll. Scenario management: An interdisciplinary approach. *Requir. Eng.*, 3(3/4):155–173, 1998.

[10] E. Nasr, L. McDermid, and G. Bernat. Eliciting and specifying requirements with use cases for embedded systems. *In Proceedings of the 7th International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'2)*, pages 350–357, January 2002.

[11] B. A. Nuseibeh and S. M. Easterbrook. Requirements engineering: A roadmap. *In A. C. W. Finkelstein (ed) "The Future of Software Engineering ". (Companion volume to the proceedings of the ICSE'00)*, 2000.

[12] J. Ralyte. Reusing scenario based approaches in requirement engineering methods: Crews method base. *In REP'99*, pages 305–309, 1999.

[13] A. Sutcliffe. Scenario-based requirements engineering. *In Proceedings of the International Conference on Requirements Engineering (RE'03)*, pages 320– 329, 2003.

[14] A. G. Sutcliffe. Scenario-based requirements analysis. *Requirements Engineering Journal*, 3(1), 1998.

[15] Y. E. Tsai, H. C. Jiau, and K.-F. Ssu. Scenario architecture - a methodology to build a global view of oo software system. In *COMPSAC*, pages 446–451, 2003.

[16] W. Wang, S. Hufnagel, P. Hsia, and S. M. Yang. Scenario driven requirements analysis method. In *Proceedings of the Second International Conference on Systems Integration*, pages 446–451, 1992.

[17] H. H. Zhang and A. Ohnishi. A transformation method of scenarios from different viewpoints. In *APSEC 2004*, pages 492–501, 2004.

PSPCAT: A PSP Data Collection and Analysis Tool

Chien-Hung Liu
Department of Computer Science and Information Engineering
National Taipei University of Technology
Taipei, Taiwan 106
Email:cliu@ntut.edu.tw

Shu-Ling Chen
Department of Management Information Technology
Southern Taiwan University
Tainan, Taiwan 701
Email:slchen@mail.stut.edu.tw

Yu-Chun Huang
Department of Computer Science and Information Engineering
National Taipei University of Technology
Taipei, Taiwan 106
Email: s2598023@ntut.edu.tw

***Abstract*— The discipline of Personal Software Process (PSP) can help developers to understand their ability, improve their software process, and increase their productivity and software quality. However, to improve the software process, developers need to gather their own process data. This does impose substantial overhead to developers. This paper presents an Eclipse plug-in tool that can help developers to measure and analyze their software process based on the principles of PSP. Specifically, the program size, process time, and defect data can be collected automatically or semi-automatically using the tool. In addition, the tool can provide valuable process information, such as the size and time statistics, defect distribution, and productivity, to help developers to assess their performance and to improve their software process.**

I. Introduction

The Personal Software Process (PSP) [1], [2] was proposed by Watt S. Humphrey to help developers to improve their personal software process. The main idea of the PSP is to understand and improve software process through planning, tracking, measuring, and analyzing the defined process. It has been shown that, by adopting the PSP practices, developers can significantly reduce the number of defects, increase the software quality and productivity, and improve the predictability of software process [3], [4]. This facilitates developers to deliver high quality software with less development time and cost.

Although the PSP practices can help developers to improve their performance, developers must collect their own process data, such as the size of programs, the time spent on each process phase, the number of injected and removed defects, the types of defects, and the defect fix time, to evaluate their process. However, gathering these process data can be time-consuming and error-prone even though the PSP framework has provided many well-designed forms to help developers recording their process data. The data gathering still involves lots of work and requires to be done consistently and carefully, which imposes a large amount of overhead to developers. Thus, a supporting tool that can help to reduce the overhead, avoid the human errors introduced in the data collection, and facilitate the process analysis has become critical for developers to use PSP.

Although there have been several PSP tools [5], [6], [7], [8], [9] that can support PSP data collection, most of the tools are not integrated with an Integrated Development Environment (IDE) tool. This makes developers have to use a PSP tool, such as a spreadsheet, to manually record their process data while using an IDE tool to develop their programs. The separation of data gathering and program development tools causes inconvenience and additional tool-switching overhead to developers. It also increases the possibility of making mistakes in recording the process data.

This paper proposes a PSP supporting tool for Eclipse platform [10], called the PSP data Collection and Analysis Tool (PSPCAT), to assist developers to collect their PSP data automatically or semi-automatically in the development phase. Eclipse is an open source programming platform and has been widely used in software community. In particular, Eclipse provides a Plug-in Development Environment (PDE) [11] that can enable plug-in tools to be integrated with Eclipse and to access the resources of Eclipse workbench. With the PSPCAT, developers can write programs and gather their process data using Eclipse without switching tools so that the data collection overhead can be reduced and the data accuracy can be improved.

The PSP can help developers to improve their software process. However, not all the developers are familiar with the PSP framework and follow the PSP defined process. In order to support most developers to gather their process data, the PSPCAT is designed to provide three data collection modes: automatic, semi-automatic, and manual modes with different levels of data accuracy. Developers can choose one of them according to their preferences or familiarity with the PSP. Moreover, to gather the defect data automatically, the PSPCAT can extract the compiling and testing error messages from the Eclipse JDT compiler and JUnit [12] and compute defect fix time automatically based on how the defects are selected to remove by the developers. In addition, through analyzing the accumulated historical data, the PSPCAT can provide various process analysis reports, such as the project summary, the trends of quality and productivity, and the statistics of defects, time, and program size. These reports can help developers to understand their current process performance and analyze the causes to improve their software process.

The rest of the paper is organized as follows. In the next section, existing PSP supporting tools are briefly reviewed. Sections 3 and 4 describe the approaches used by PSPCAT to gather the PSP time and defect data, respectively. Section 5

presents the architecture design and the uses of the PSPCAT. The last section summarizes the conclusions and describes our future work.

II. Related Work

In this section, we briefly review several PSP data collection and analysis tools and make a comparison between these tools and PSPCAT.

Personal Software Process Studio (PSPS) [5] is proposed to facilitate the PSP discipline. It supports the tables and forms required for the entire PSP framework from PSP0 to PSP3. Basically, the PSPS allows developers to manually record the program size, time, and defect data. In addition, it provides PSP size estimation and project planning forms, design templates, project summary report, and Process Improvement Proposal (PIP) tables. The major deficiency of the PSPS is that developers need to keep track of the process data by themselves and have to manually record the data into the PSPS.

PSP Process Dashboard [6] is an open source PSP tool that provides all the scripts, tables, forms, standards, and analysis reports defined in the PSP. The PSP Process Dashboard completely advocates the PSP framework. It supports all the process definitions from PSP0 to PSP3 and, for each process definition, it includes all the process phases from the planning to the postmortem phase. However, similar to the PSPS, the PSP Process Dashboard is a standalone tool which is not integrated with any IDE tool. Thus, it requires developers to manually record their process data.

Hackystat [7] is an open source framework for automatic collection and analysis of process and product data. The Hackystat employs a client-server architecture. In particular, the Hackystat has provided various clients (or sensors) integrating with other tools, such as Eclipse, Emacs, JUnit, and Ant. These sensors can gather process and product data and send to a repository at server for further analysis. Developers can check their process and product data and corresponding analysis from the Hackystat server. Hackystat is able to collect the PSP data automatically. However, many defect data, such as defect type, defect injection and removal phases, and fix time, are not gathered. This could pose some limitations on defect analysis.

PSP Assistant (PSPA) [8] is a client-sever PSP supporting tool. The client of PSPA is an Eclipse plug-in that can automatically gather the program size and some defect data. Specifically, the PSPA can classify and rank the defects to assist developers to analyze their performance. It also can consolidate the schedules and defect data of individual developers into a schedule and defect library for the development team.

DuoTracker [9] is a tool for supporting software defect data collection and analysis. The DuoTracker can be integrated into a defect tracking tool, such as Bugzilla [13]. It allows developers to view the defect records reported in the defect tracking tool. The developers then can enter the PSP required information for selected defects. In addition, by integrating with the defect tracking tool, the DuoTracker can gather the defects occurred beyond the compile and test phases.

Table I shows the comparisons of the PSPCAT and other PSP supporting tools. In particular, the PSPCAT mainly focuses on the development phases of the process. It can avoid the cumbersome introduced by tool-switching and gather PSP data automatically or manually with different levels of data accuracy. Moreover, the PSPCAT can automatically compute the defect fix time and suggest the defect type. It also provides several PSP analysis reports and can export the process data in terms of XML format for the consolidation of team process data.

III. The Collection of the PSP Time Data

In order to better understand and improve software process, the PSP framework explicitly defines the process structure. Basically, a defined process consists of several phases, such as plan, design, code, compile, test, and postmortem. Developers have to gather the time and defect data for each phase. Since most developers employ the IDE tool only for developing programs not for planning or designing the projects, the PSPCAT is designed to gather the process data for the code, compile, and test phases instead of the entire PSP process.

Because not all the programmers are familiar with the PSP framework, the PSPCAT provides three data gathering modes: automatic, semi-automatic, and manual modes with different levels of data accuracy and user intervention to support most programmers to gather their process data. Developers can choose the data collection mode according to their familiarity with PSP or their needs for data accuracy. For example, the automatic mode can automatically track the process phases and gather the time and defect data of each phase without user intervention. However, this mode is unable to count the interrupt time of the process since there is no way to determine if the developers are thinking of the design or taking a break when they are not interacting with the computer. As a result, the time data gathered may not be fully accurate. To count the actual time spent on each phase correctly, developers can manually pause and resume the PSPCAT timer for each break (manual mode) or they can manually modify the automatically recorded time data to reflect the interrupt time (semi-automatic mode).

Moreover, the Eclipse supports continuous compilation in order to save the compilation time. This means that, when developers are writing code, the Eclipse will continuously compile the programs and show the syntax errors found so far. In such a case, there is no specific compile phase as defined in the PSP process structure. However, if the developers would like to analyze their process in detail, they may want to disable the continuous compilation in the Eclipse and to get their time spent on the compile phase. To support this, the PSPCAT gathers the time data of compile phase in the semi-automatic and manual modes. Table II lists the differences among the three data collection modes supported by PSPCAT. Basically, the automatic mode can change the process phase automatically, but it does not consider the compile phase and interrupt time. The manual mode requires developers to switch the process phases and record the interrupt time manually. The

TABLE I

THE COMPARISONS OF PSPCAT AND OTHER PSP TOOLS

Name	Time Logging	Defect Logging	LOC Counting	Project Summary/Analysis Reports	Data Export
PSP Process Dashboard	manual	manual	manual	project plan summary (PPS), analysis (full support)	no
PSPS	manual	manual	manual	PPS, analysis (full support)	no
Hackystat	auto	auto (no defect type & fix time)	auto	process summaries, analysis	yes
PSPA	auto	auto (no defect type & fix time)	auto	PPS, analysis (schedule, defects)	yes
DuoTracker	N/A	semi-auto	N/A	N/A	no
PSPCAT	manual/auto	manual/auto	manual/auto	PPS, analysis (quality, time, defect, size)	yes

TABLE II

THE DIFFERENCES AMONG THE PSPCAT DATA COLLECTION MODES

Differences	Auto Mode	Manual Mode	Semi-Auto Mode
Supported Process Phases	code, test	code, compile, test	code, compile, test
Change of Phase	automatic	manual	automatic (allow manual change)
Time Calculation	automatic (no interrupt time)	automatic (allow to pause/resume timer & change data)	automatic (allow to pause/resume timer & change data)
Interrupt or Break	not support	support	support
Defect Collection	automatic (no compile phase)	automatic	automatic
Cyclic Programming (class level)	support	support	support
Process Analysis Reports	productivity, quality, defect, time, size (no compile phase)	productivity, quality, defect, time, size	productivity, quality, defect, time, size

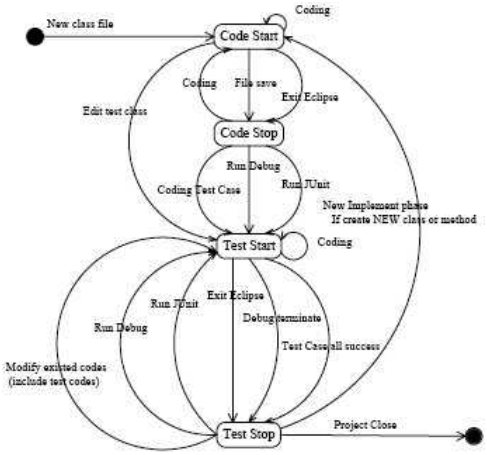


Fig. 1. The state diagram used to track the process phases (automatic mode)

semi-automatic mode is similar to the automatic mode, except it allows the recorded data to be updated manually.

It should be noted that the developers may not exactly follow the order of process steps defined in the PSP framework which requires the process phases to be changed sequentially. For example, developers can finish all the coding first and then compile and test the program; or they can code a little and test a little. To count the time spent on each phase without user involvement (i.e., automatic mode), the PSPCAT has to automatically track the current process phase of the project under development. To achieve this goal, the PSPCAT considers the start and stop of coding, compile, and test tasks as different states. The various events invoked by the developers or JUnit, such as creating a new class, exiting Eclipse, and invoking JUnit, are considered as transitions between the states. By listening to these events and computing the time period between the start and stop of the tasks, the PSPCAT can obtain the time that developers spend on each task (i.e., phase).

Figure 1 shows the task state diagram used to track the process phases in the automatic mode of the PSPCAT. The PSPCAT will detect the events occurred and change the states accordingly. For example, the occurrence of new class event indicates that the coding task is started. Likewise, the occurrence of "file save" event suggests that the coding task is stopped. Thus, the time spent on the code phase can be obtained by computing the time period between the start and stop of the coding task. Similarly, the process will change from the code phase to the test phase when developers create a new test class, lunch the Eclipse debug function, or invoke JUnit. The process will remain at test phase until the Eclipse debug function is terminated or all the JUnit test cases are passed.

IV. THE COLLECTION OF THE PSP DEFECT DATA

As compared with the time data collection, the gathering of the defect data is more time-consuming and error prone. Thus, the major focus of PSPCAT is to reduce the defect data collection overhead while increasing the data quality. To reduce human involvement, the PSPCAT employs two techniques to obtain the defect data automatically through (1) inserting a listener at Eclipse workbench to observe the compiling error messages; and (2) using the extension points provided by JUnit to obtain the execution results of test cases. The former allows the PSPCAT to gather the compiling defect data in the compile and test phases and the later allows the PSPCAT to collect the functional defect data in the test phase.

Note that the PSP defect data include the time when the defect was found, the defect number, the defect type, the defect injection phase, the defect removal phase, the defect fix time, and a brief defect description. Based on whether the defect data can be automatically obtained, the PSPCAT either collects the data directly or simply gives a default value. For example, to gather the defect descriptions, the PSPCAT obtains

the error messages directly from the Java compiler and JUnit. For gathering the defect types, the PSPCAT will initially assign the type of syntax error to those defects found by the compiler and the type of function error to those defects revealed by the JUnit. After reviewing the defect, developers can change the initial value and provide an appropriate defect type for the defect if necessary.

Moreover, to compute the defect fix time automatically, the PSPCAT exploits the "Goto" function provided by Eclipse. The "Goto" function basically allows developers to double click on an error message appeared in the Eclipse problems view so that the cursor of the Java editor will automatically move to the location of the corresponding error. It can help to locate the defect that developers would like to fix. Thus, by listening to the "Goto" event, the PSPCAT can obtain the defect that the developers may possibly aim to fix and record the start time (denoted as T_{start}) for removing the defect. After changing and recompiling the code, if the error does not appear in the problems view, the PSPCAT will consider this error being removed and record the stop time (denoted as T_{stop}) for fixing the defect. Thus, the defect fix time (denoted as T_{fix}) for this error can be computed using the following equation:

$$T_{fix} = (T_{stop} - T_{start})/N, \tag{1}$$

where N is the number of defects disappeared in the problems view due to this code modification. Since developers may not use the "Goto" function to fix a defect, the PSPCAT considers three possible debugging practices: (1) All defects are removed using the "Goto" function; (2) No defects are removed using the "Goto" function; and (3) Only part of the defects is removed using the "Goto" function. For the first case, the defect fix time can be computed by simply using equation (1). For the second case, the PSPCAT does not know which defect is going to be fixed by developers. Under such circumstance, the PSPCAT considers the defect fix time of each individual error as the average time spent on fixing a defect. Assume that the process is in the compile phase, the total number of compiling errors found is N_{total}, and the time spent on the compile phase is $T_{compile}$. The average fix time for each compiling error can be obtained using the equation below:

$$T_{fix} = T_{compile}/N_{total}. \tag{2}$$

For the last case, the PSPCAT uses equation (1) to compute the fix time for those defects removed through using the "Goto" function. The fix time for the remaining defects is derived from equation (2). Suppose that the process is in the compile phase, the total number of defects removed by the "Goto" function is N_{goto}, and the total time to fix these defects is T_{goto}. The average fix time for the remaining defects then can be derived from the following equation:

$$T_{fix} = (T_{compile} - T_{goto})/(N_{total} - N_{goto}). \tag{3}$$

In addition to compute the fix time of compiling defects, the PSPCAT also derives the fix time for the functional defects appeared in the JUnit failures window. The computation of fix time for the functional defects is similar to that of fix time for the compiling defects since the JUnit also supports the "Goto" function allowing developers to double click on a failure test method in order to open the test method in the Java editor.

V. The Architecture Design and the Uses of the PSPCAT

In this section, the system architecture of PSPCAT is briefly described and some screen shots are provided to illustrate the uses of the PSPCAT. Figure 2 shows the architecture design of the PSPCAT. Basically, the PSPCAT consists of four subsystems. Each of the subsystems is described briefly as follows:

- The Data Collection Subsystem (DCS) mainly focuses on the gathering of process data, including the project plan summary, size, time, and various defect data.
- The Controller and Plug-in Subsystem (CPS) manages the interactions among the subsystems and the Eclipse platform. In particular, the CPS controls the changes of the PSP phases, observes various Eclipse events through the workbench, and accesses the project information through the Eclipse workspace.
- The Data Analysis Subsystem (DAS) provides the PSP project plan summary and various statistic analysis reports based on the collected process data, such as the size and time trends, productivity, defect distribution, and quality analysis.
- The Data Management Subsystem (DMS) mainly supports the management of the gathered process data. It provides the repository for the historical project data and the functionality to export and upload the data to external server.

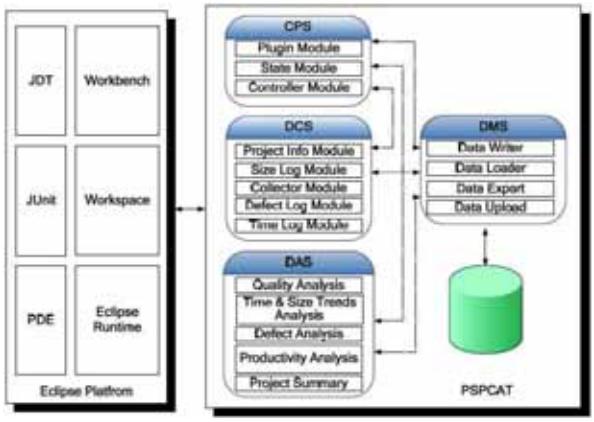


Fig. 2. The architecture of the PSPCAT

To use the PSPCAT, developers first need to enable the data collection and specify the data collection mode and data repository. Once the data collection is enabled, the process

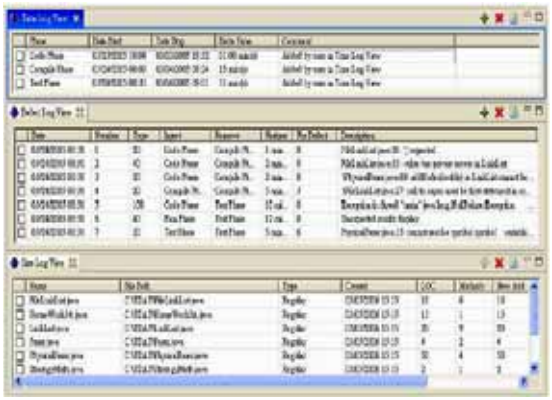


Fig. 3. Examples of the size, time, and defect logs in PSPCAT

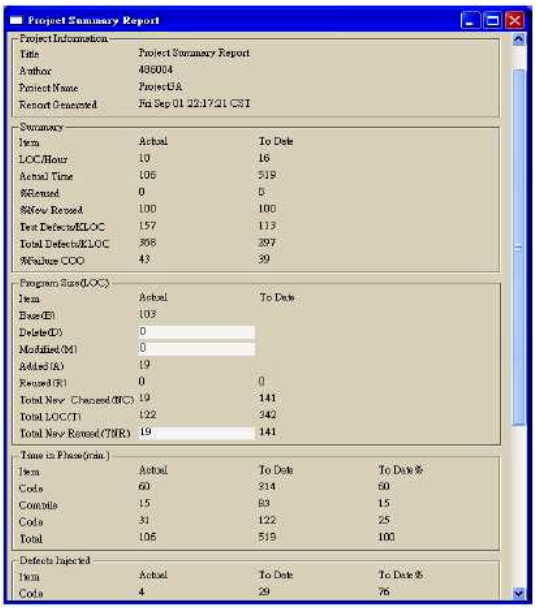


Fig. 4. An example of the PSPCAT project plan summary

data of the project under development can be gathered automatically or manually. The gathered data are then shown in the corresponding log views. Figure 3 presents the log views for the size, time, and defect data. Developers can insert, delete, update, or sort out the data records in each log view.

After a project is completed, the PSPCAT can analyze the process data logs and provide the PSP plan summary for the project. As shown in Figure 4, the project plan summary includes the overall summary of the project and the statistics of the collected size, time, and defect data. Based on the gathered data, the PSPCAT also provides process analysis reports that can help developers to examine and improve their software process. Figure 5 shows some examples of the process analysis reports provided by PSPCAT.

VI. Conclusions and Future Work

This paper has presented an Eclipse plug-in tool called PSPCAT to support automatic PSP data collection and analysis. By integrating with the Eclipse platform, the proposed tool

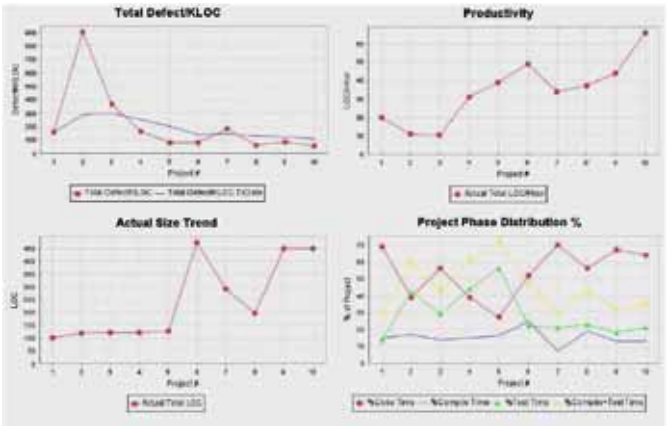


Fig. 5. Examples of process analysis reports provided by PSPCAT

can reduce the tool-switching overhead and the possibility of making mistakes in gathering process data. In particular, the tool provides three data collection modes with different levels of data accuracy to support different needs of developers and programming practices. The proposed tool also provides various analysis reports to help developers to understand their development performance and to improve their software process.

Currently, the PSPCAT gathers only the PSP data in the code, compile, and test phases. In the future, we plan to enhance the PSPCAT to support the entire PSP framework. Moreover, we plan to extend the PSPCAT to support the Team Software Process (TSP) [14] based on the client-server architecture. The process data of each team member collected by the PSPCAT client can be consolidated at the PSPCAT server so as to obtain the TSP data to analyze the team performance and to improve the team software process.

References

[1] W. S. Humphrey, A Discipline for Software Engineering, Addison Wesley, 1995.
[2] Personal Software Process (PSP). http://www.sei.cmu.edu/tsp/psp.html
[3] P. Johnson and A. Disney, "The Personal Software Process: A Cautionary Case Study," IEEE Software, Vol. 15, No. 6, November, 1998, pp. 85-88.
[4] J. Kamatar and W. Hayes, "An Experience Report on the Personal Software Process," IEEE Software, Vol. 17, No. 6, 2000, pp. 85-89.
[5] Personal Software Process Studio (PSPS). http://www.cs.umt.edu/RTSL/dsstud/psp/psps.htm
[6] Software Process Dashboard Initiative. http://processdash.sourceforge.net/
[7] Hackystat. http://code.google.com/p/hackystat/
[8] R. Sison, D. Diaz, E. Lam, D. Navarro, and J. Navarro, "Personal Software Process (PSP) Assistant," Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05), 2005, pp. 687-696.
[9] O. Akinwale, S. Dascalu, and M. Karam, "DuoTracker: Tool Support for Software Defect Data Collection and Analysis," Proceedings of the International Conference on Software Engineering Advances, 2006.
[10] Eclipse. http://www.eclipse.org
[11] PDE. http://www.eclipse.org/pde/
[12] JUnit. http://www.junit.org
[13] Bugzilla. http://www.bugzilla.org/
[14] TSP. http://www.sei.cmu.edu/tsp/tsp.html

A Systematic Method for Process Tailoring Based on Knowledge Reuse

Xiao-yang He, Ya-sha Wang*, Yu-xin Teng and Jin-gang Guo
Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China;
Key laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China

Abstract Process tailoring largely depends on experts' knowledge and might be time-consuming, costly and error-prone. Actually, process tailoring knowledge should be captured, documented and reused for future projects. To address this problem, a systematic method for process tailoring is proposed. The factors that strongly influence process tailoring are recognized as process drivers. The reusable process tailoring knowledge is organized by packages. Each package contains a certain process driver and its impact on the process framework which is the general process for tailoring. Aided by a conflict resolving model which checks and resolves possible conflicts between different packages, the packages can be reused for generating a project-specific process. A case study of RUP tailoring is presented to validate the method proposed.

1. Introduction

A well-defined software process is critical to improving software productivity and quality. Many a software process framework has been published, such as Rational Unified Process(RUP)[1], V-Modell XT[2] etc., which provides a good starting point in software process construction. However, they have to be tailored, downscaled and specialized to the context of use[3, 4].

Certain factors that strongly influence process tailoring have been recognized, for example, system scale, system complexity and time pressure of the project. Generally, experts make tailoring decisions based on the perceiving of the factors in a given context and the knowledge of the factors' influence on the process. These factors are named as process drivers in this paper.

According to [3], the practice of adjusting a standard software development process to accommodate differences among projects is called tailoring. So far, process tailoring largely depends on experts' knowledge, and might be time-consuming, costly, and error-prone[4, 5]. To address this problem, a systematic way for process tailoring based on knowledge reuse is proposed.

Our method consists of two steps. (1) Capturing and documenting the reusable knowledge of process tailoring. The reusable knowledge is packaged in the light of process drivers. Each package contains the information of a certain process driver and its influence on the process. (2) Tailoring the process by reusing the packaged knowledge. A conflict resolving model, which is abbreviated to conflict model, give support to resolve possible conflicts when more than one package is applied to process framework.

2. The Method for Process Tailoring

The purpose of this paper is to improve the effectiveness and efficiency of process tailoring by capturing and sharing the tailoring knowledge within the organization. The knowledge is explicitly documented and organized as reusable process tailoring packages.

There are two principal roles. The first one is Process Engineer who is in charge of capturing the process tailoring knowledge and developing the tailoring packages for reuse in different projects, while the other one is Project Manager who is responsible for creating project-specific process with reuse of the tailoring packages. The process of the method is presented with UML activity diagram in Figure 1.

Generally, Process Engineer has abundant knowledge of the process framework, as well as the insight into the development situation of the organization. He is responsible for identifying process drivers from different aspects (e.g. project, organization, person and product). These drivers characterize the profile of development situation, and each driver defines some project-specific requirement of the process which can be transformed into process tailoring operations on the process framework. For example, the *new system development* project requires more user-developer interaction than the *existing system modification* project. The Process Engineer may identify a process driver, which is called *new system development*. This process driver requires supplementing some user interaction activities and

The work is funded by Grant 2006AA01Z189 from Hi-Tech Research and Development Program of China and Grant2005CB321805 from the National Key Basic Research of China.
*Corresponding author

adding additional prototype-development iteration into the process framework; furthermore, the activity of *analyze the difference between the existing system and new system*

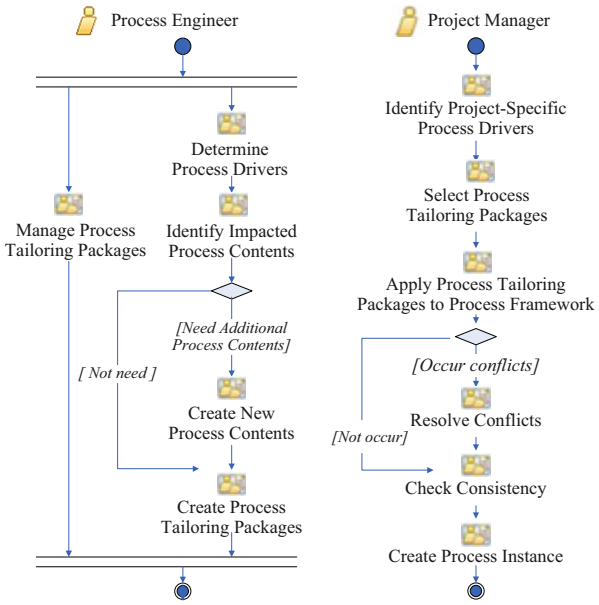


Figure 1 Process of the Tailoring Method

requirement in the process framework can be deleted. For every process driver, Process Engineer should analyze which process contents are impacted. If the existing process could not fully meet with the requirements of process drivers, new process contents are created. Process Engineer, then, creates Process Tailoring Packages, which present how process drivers impact the process contents. Process Tailoring Package itself is not a complete software process but externally defines the modifications to process framework.

Project Manager analyzes the characteristics of the project at hand and determines the project-specific process drivers. For every process driver, a Process Tailoring Package is selected. When more than one package is applied to the process framework, conflicts may occur. In this case, the project manager should resolve the conflicts with the help of a predefined conflict model explained in section 4. Finally, the project-specific process instance is created.

3. Meta-model of Reusable Knowledge Packages

To effectively represent and organize the process tailoring knowledge, a meta-model is proposed to demonstrate the core concepts and their relationships. It is shown in Figure 2 with UML class diagram.

For the sake of concision and clarity, we only focus on those fundamental concepts of process meta-model. A Work Definition describes a unit of work with the clear purpose, usually expressed in terms of creating or updating some Work Products. Within a Work Definition, each performing Role achieves a well-defined goal. Work Definition is the abstract class of Activity and Task. Activity supports the nesting and logical grouping of Tasks which can be broken down into Steps.

A Process Driver focuses on a certain concern which the process tailoring aims to handle. Every Process Driver is associated with a Process Tailoring Package. A Process Tailoring Package consists of one to more Impact Pieces which describes the information of exactly one tailoring operation. There are three classes named Contribute, Suppress, and Replace implementing the abstract class Impact Operation. They define the concrete types of tailoring operation, which denote adding, deleting and modifying Work Definition respectively.

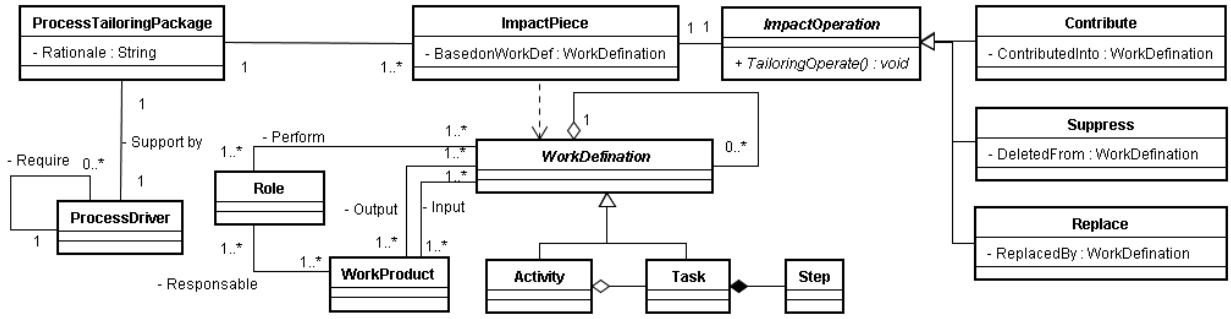


Figure 2 Meta-model of the Reusable Tailoring Knowledge

4. Conflict Resolution

Conflicts may occur when more than one Process Tailoring Package is applied to process framework. The conflicts result from different Process Tailoring Packages attempting to modify the same Work Definition in different way. For example, one Process Tailoring Package may attempt to replace an activity while another one may want to remove it.

In order to resolve the conflicts, two issues should be concerned: (1) How to identify the place where the conflict occurs? (2) What action should be taken if conflict occurs?

According to the place where the conflicts occur, two types of conflict are identified. Assuming Work Definition W_a and W_b are impacted by two Process Tailoring Packages, say package P_a and P_b respectively. If W_a and W_b are exactly the same Work Definition, a homo-place conflict

occurs. If W_a is a part of W_b (e.g., W_a is a sub-activity of W_b), a hetero-place conflict occurs, vice versa. Considering an example, a certain package demands delete an activity, while another package requires replace a task in the same activity, a hetero-place conflict appears.

+C	*+C*		
+S	*N*	*+S*	
+R	*N*	*+S/+R*	*H*
	+C	**+S**	**+R**

(a) Homo-place Model

+C	*N*	*N*	*N*
+S	*+S/-C*	*+S*	*+S/-R*
+R	*+R*	*+R*	*+R*
	- C	**- S**	**- R**

(b) Hetero-place Model

Figure 3 Confliction Model

The conflict models of these two types are shown in Figure 3. C, S and R respectively represent the tailoring operation Contribute, Suppress and Replace. Additionally, a special mark ('+' / '-') is added as the prefix of the Impact Operation. Symbol '+' means the Impact Operation directly acting on the Work Definition; Symbol '-' means the Impact Operation acting on the sub-element of the Work Definition. The cross units represents the action of conflict resolution actually taken. Besides the typical Impact Operation types, there are additional action types:

- *N*: Two operation types would not occur simultaneously.
- *H*: Operation actually adopted is up to Project Engineer.

Moreover, the reasons behind the operation 'S' are further analyzed:

- *Not needed*: the Work Definition is not needed at all. An example is that the development of operation system does not need the activity of "Design the Database".
- *Valueless*: There is not enough time and effort to implement a certain Work Definition, or the Work Definition would not bring any value.

Symbol '/' separates the action types by the reasons behind the operation 'S'. The former action is taken if the reason is "Not needed", or else the latter is taken.

```
Create a tree T by composition relationship between the Work Definitions;
For every ImpactPiece instance I:
   mark node I.BasedonWorkDef with '+' and I. OperationType;
Post-order Traversal (T):
   While node X.Number of marks >=2 and X.Number of marks ('+')>=1:
      Select any Impact Operations Ia, Ib on condition that (Ia.mark = '+'
         or Ib.mark = '+)';
      If Ia.mark='+' and Ib.mark='+':
         Then take action A according to Figure 3(a);
      Else take action A according to Figure 3(b);
      If A ='H':
         Then { Insert node X into Conflict list C; set X.OperationType='R';}
      Delete unneeded Operation Ia or Ib from Node X;
   End while;
   If X.OperationType ='S': Then delete node X and all X Offspring;
   Else if X.OperationType ='R': Then delete all node X.Offspring;
   If X. Number of OperationType >0:
      Then Append mark '-' and X. OperationType to X.Parent
End Traversal;
If C.IsEmpty: output (T);                    //Output the tailored process
Else output C;          //Output all conflicts needed manual intervention
```

Figure4 Conflict Checking and Resolving Algorithm

A conflict checking and resolving algorithm described with pseudo code is shown in Figure 4. Firstly, a tree is generated in terms of composition relationship between the Work Definitions in process framework. Special marks are added to corresponding node for every Impact Piece. Each node is checked by post-order traversal then. If there is a homo-place or hetero-place conflict, an action is taken according to conflict model. If there is no conflict which needs manual intervention, a tailored process is generated, or else the places conflicts occur are pointed out.

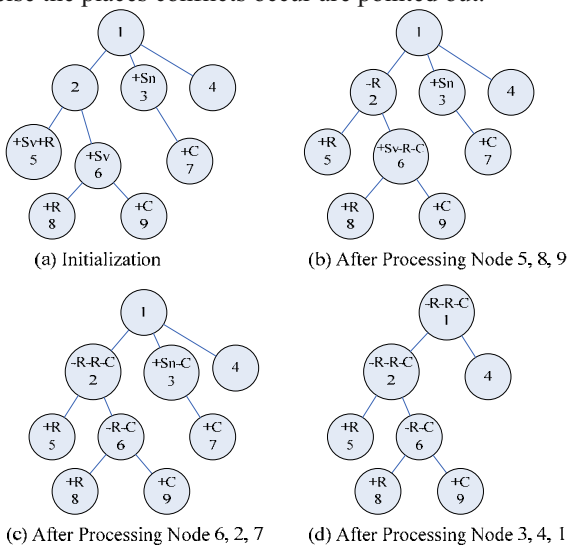


Figure 5 Example of the Algorithm Applied

An example is shown to illustrate the algorithm mentioned above. The initial status of the tree is shown in Figure 5(a). The nodes are checked in turns by post-order transversal. In Figure 5(b), the Impact Operation 'R' of node 8 and the Impact Operation 'C' of node 9 are transferred to node 6 and marked with '-', so the mark of node 6 is changed to "+S-R-C". There is a hetero-place conflict between "+S" and "-C", thus the conflict is resolved according to Figure 3(b). In particular, the operation 'S' here is "Valueless" suppression, so the actual tailoring operation adopted is 'R' and the mark of node 6 is changed to "-R-C". Now node 6 doesn't have any other conflict and its operation 'R' and 'C' is transferred to node 2. Other nodes are processed in the same way, shown in Figure 5(c) and 5(d).

5. Case Study

For empirical validation of the method proposed, we applied it in a company which specialized in software development of web-based solutions to general business. The company had adopted RUP as its process framework but process tailoring was carried out in an ad-hoc way, so it showed great readiness to improve process tailoring.

First, we worked with the company's experts to identify those distinct characteristics of the organization and its common project types. After that we captured and documented the knowledge of how these drivers impact the RUP based on the experts' previous experience. For

example, the driver of "High Usability" specially focuses on analyzing user and usability requirements, and asks for adding two new tasks into the activity of "Understand Stakeholder Needs". A process asset library is established to store the reusable process tailoring knowledge.

A project of online bookstore made a request for tailoring process. The size of project was small that only five developers were engaged within the duration of three to four months. The customers of online bookstore are members of the general public, not technical experts. Hence, application usability must be high. The customers also pay extremely attention to privacy protection and security of transaction security. As a result, the drivers of "Small Size Project", "High Usability" and "High Security" are the first-class concerns. Then the corresponding process tailoring packages selected are applied to RUP to generate project-specific process. For lack of space, the resulting process is omitted in this paper.

With the accumulation of the process tailoring knowledge within the organization, the method proposed can drastically improve the efficiency of tailoring.

6. Related Work

Although process tailoring is a mandatory activity for organization adopting process framework, it strongly depends on experts' knowledge[4]. The amount of research done on process tailoring to date can be considered small.

It is shown that knowledge reuse can improve efficiency and effectiveness of process tailoring[5]. Case-based reasoning is adopted to facilitate reusing the cases to customize the target process [6, 7]. However, any two projects are different, so a process successfully applied to one project may not work in another[4]. Partitioning the tailoring knowledge by process driver has much more reusability and flexibility than the case. The knowledge based inference technique is also utilized in [7] to derive process. Every process driver is supported or deactivated by a set of activities. The selected activities are integrated to generate process. But it is very troublesome to combine activities one by one. Its conflicts resolution is achieved by constraining selecting incompatible drivers simultaneously. Such a disposal is not reasonable in practice. Karlsson[8] utilizes the concept of Configuration Packages which are pre-made reusable configurations of a base method suitable for a characteristic. Combining configuration packages will result in overlapped activities and increase the overload of tailoring effort. He just simply adopts priority-based policy and leave human to deal with conflicts.

RUP itself contains configuration activities, which create a Development Case. But these activities are not sufficient, at least not considering reusability of tailoring knowledge.

7. Conclusions and Future Work

No single process is suitable for all software projects. The process framework must be adapted to the specific context, which requires substantial up front effort. Reuse of the tailoring knowledge can drastically improve the effectiveness and efficiency of tailoring.

This paper proposes a method for process tailoring based on the concepts of process driver and process tailoring package. Partitioning tailoring knowledge by process driver is propitious to narrow the focus at a time and improve reusability. In particular, the process tailoring package externally defines the changes to the process framework, which provides more flexibilities than directly modifying process framework. The way of applying process tailoring packages to process framework is more efficient than that of combining activities required by process drivers. Moreover, the decision model and the algorithm proposed provide the strong support for conflict checking and resolving.

The meta-model proposed in this paper is based on the Software Process Engineering Specification v1.1 (SPEM)[9]. The forthcoming SPEM v2.0 plans to separate Method Contents from Process. More dedicated analysis should be taken to deal with the new change.

Reference

1. Jacobson, I., G. Booch, J. Rumbaugh, *The Unified Software Development Process*. 1999: Addison Wesley Longman.
2. *V-Modell XT*. Available from: http://www.v-modell-xt.de/.
3. Ginsberg, M.P., L.H. Quinn., *Process Tailoring and the Software Capability Maturity Model*. 1995, Carnegie Mellon University, Software Engineering Institute.
4. Pedreira, O., M. Piattini, M.R. Luaces, et al., *A Systematic Review of Software Process Tailoring.* ACM SIGSOFT Software Engineering Notes 2007. **32**(3):1-6.
5. Peng, X. *Knowledge Support in Software Process Tailoring*. in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*. 2005. 87-95
6. Henninger, S., K. Baumgarten. *A Case-Based Approach to Tailoring Software Processes*. in *Proceedings of the 4th International Conference on Case-Based Reasoning*. 2001. 249-262
7. Ahn, Y.W., H.J. Ahn, S.J. Park, *Knowledge and Case-Based Reasoning for Customization of Software Processes: A Hybrid Approach.* International Journal of Software Engineering and Knowledge Engineering, 2003. **13**(3): 293-312.
8. Karlsson, F., P.J. Agerfalk, *Method configuration: adapting to situational characteristics while creating reusable assets.* Information and Software Technology, 2004. **46**(2004):619-633.
9. OMG, *Software Process Engineering Metamodel Specification, Version 1.1*. 2005.

Linking Return on Training Investment with Defects Causal Analysis

Santiago Matalonga
Universidad ORT Uruguay
smatalonga@uni.ort.edu.uy

Tomás San Feliu Gilabert
Universidad Politécnica de Madrid
tsanfe@mpsei.fi.upm.es

Abstract

In this paper, we present a process for linking organizational training efforts with defects causal analysis in software development organizations. The process is being implemented in a CMMI maturity level 3 organization. Since causal analysis is not an expected process area at maturity level three, key success factors for the implementation of the process are identified and analyzed. The conclusions were tested in this software development organization. In order to do that, a pilot project was selected and training was implemented to support the process. The training results are analyzed in order to validate the overall approach. The resulting work provides a guideline for implementing causal analysis in lower maturity organizations and establishes that the implementation is viable in the target organization.

1. Introduction

Organizations nowadays invest large amounts of money in their training programs. Just in the United States, organizations invest an average of U$S 100 billion annually [1]. Organizations invest in training under the assumption that higher trained employees will result in higher quality products and reduced costs.

Nevertheless, in spite of all this investment and effort, organizations have not been so successful at providing consistent data that will link the investment on organizational training to organizational results.

Kirkpatrick's four levels of training evaluation establishes a framework against which organizations can measure up its investment in training. Kirkpatrick's four levels are called: *Reaction,* where trainees feelings towards the training are measured; *learning,* were trainees acquired knowledge is measured; *Transfer,* in which a measure of the amount of the Learned knowledge as actually put to use in the work; and *Results,* were impact to the organization's bottom line results is measured.

In this paper we propose a process for an organizational training department within a CMMI [2] maturity level 3 (CMMI L3), that addresses this problem. Our objective is to design a training process that is able to present results at Kirkpatrick's "Results" level [3].

The key aspect for the successful implementation of our process is the capability of the organization in defects causal analysis. Since the Causal Analysis and Resolution (CAR) process area belongs to maturity level 5 in the staged representation of the CMMI model, we want to make sure that the implementation of some causal analysis specific practices is possible in a level 3 organization.

In the following section, we provide an overview of the target organization, followed by our process proposal for the training department. Then in section 4, , we provide an analysis of the current state of causal analysis research. We specifically focus on linking causal analysis to organizational training. We finally provide data on the implementation and validation of causal analysis sessions within a maturity level 3 organization.

2. Brief description of the target organization and its infrastructure

The organization we are working with has recently obtained a maturity level 3 rating. It is a software factory that provides customized software solutions to in-shore and off-shore customers. Its development offices are located in Uruguay, and it has sales offices in the Caribbean and Mexico.

In the past two years, the organizational training department has invested - in US dollars - the equivalent to 7% of billable working time of their software developers. As a result, the training department needs to show the organization the return of its investment.

A Microsoft Office SharePoint Server supports the organizational measurement system. The SharePoint Server allows for the interoperation with other Office tools, for instance Microsoft Access is used for data analysis needs and the front end for data recollection is usually a Microsoft InfoPath form. The organization defect tracking system is one example.

Currently, the training department is using the described tools for its measurement needs. However, its data repository is not yet linked to the defect tracking system.

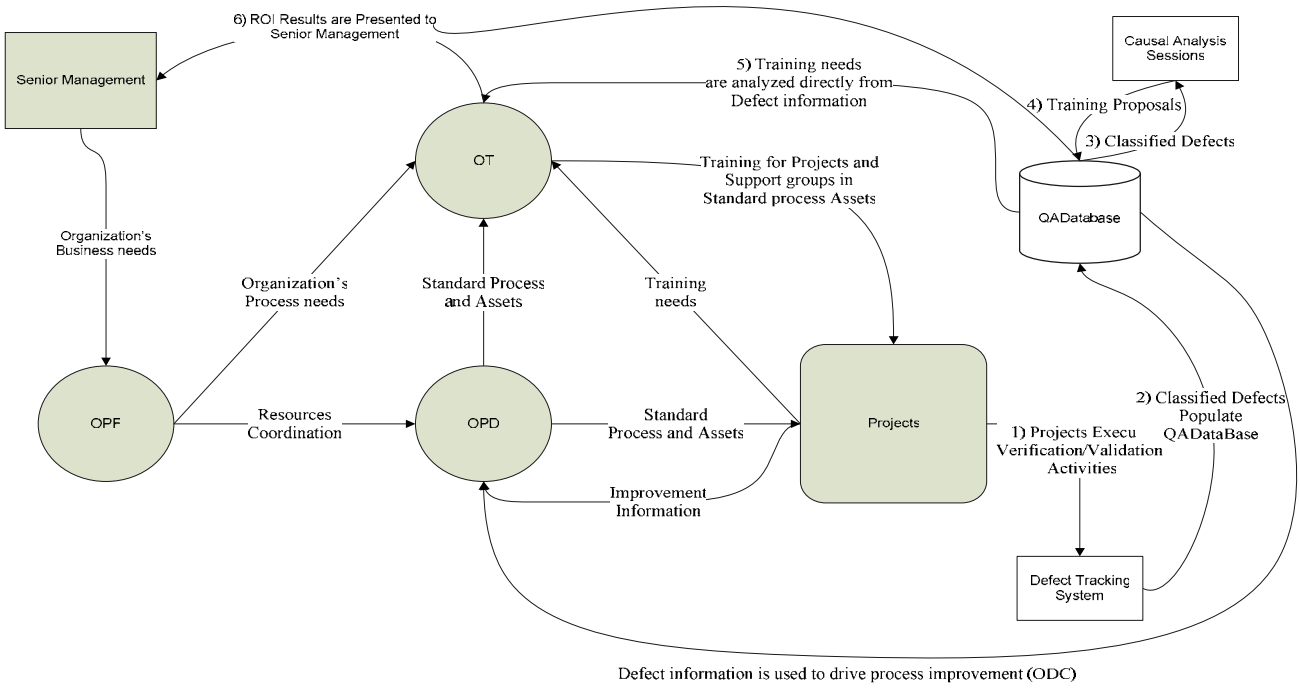


Figure 1 Overview of the Proposed Process

3. A description of our proposed process

This section describes the process we are deploying in order to link organizational training results to defects causal analysis. The idea is to make use of the information available at a CMMI L3 organization. Specifically, we want the training department to take advantage of the data available in the organization's defect tracking system. The purpose is that the training department will monitor the defect data in the system and will interpret the data as the training needs of the organization's development projects. The training department will use this information to plan training interventions specifically tailored to the development projects needs. We expect that those interventions will have positive results in the projects quality, and that those results will be noticeable in a reduction of the number of defects.

Figure 1 uses the diagram from the CMMI basic process management areas [2] in order to help us illustrate our process (items which come from the original CMMI diagram are shown in grey color, items from our proposal are left in white).

The first requirement of our process is that the defect tracking system must support a classification scheme that will allow for easier data analysis. IBM's Orthogonal Defect Classification (ODC) [4] is an example of one of such schemes, and the one we have used as reference. ODC is described in section 5.1.

In this process, when developers execute Verification and Validation activities they are required to classify the defects they log into the defect tracking system (arrow **1**). During the course of the projects life cycle, the execution of Validation and Verification activities will populate the Defect Tracking System database (QADataBase) with classified defects (arrow **2**).

At every project's milestone, the development team (arrows **3**) holds causal analysis meetings. The objective of these meetings is to provide the training department with training proposals. The idea is that developers will take the classified defect data (arrow **3**) from the QADatabase and identify which training they would have needed in order to prevent some of the mistakes that triggered the defects (arrow **4**).

The training department will use both the classified defects from the QADatabase and the outputs from the causal analysis sessions in order to plan the training interventions (arrow **5**).

Arrow **6** represents the ultimate goal of this proposal, which is the ability to show results in terms of Return on Investment (ROI) [5]. Return on Investment is interpreted as a Kirkpatrick's "Results" level measurement. Return on investment can be calculated by taking into account the defect reduction that should be noticed after the training department intervention. We expect to use the ROI classical formula to calculate ROI:

$$ROI = \frac{100 * (Benefits - Costs)}{Costs}$$ [5].

As a first step into the implementation of this process, we will turn our attention to validate arrows **3** and **4** of this cycle. First, we will present a systematic review of available literature about the implementation of causal analysis process area in lower maturity organizations. Our objective is to see if there have been previous attempts at

implementing causal analysis in a context similar to the one we have described. If the review is successful, we will turn our attention into developing and validating a consistent classification scheme within the organization.

4. A review of causal analysis methodologies

Since Causal Analysis and Resolution (CAR) is a CMMI L5 process area, we cannot expect to see a CAR process deployed in a level 3 organization. Yet, the causal analysis meeting is the critical success factor of our process since the output of the causal analysis meeting (arrow **3**) is one of the inputs that the training department will need in order to plan and deploy the training interventions.

A systematic review [6] of the causal analysis bibliography was carried out. Our objective was to answer these two questions: **A)** Has anybody else tried to implement CAR at a CMMI L3 organization? **B)** What are people doing with the results of the CAR meeting?

Table 1 shows a summary of the results of the systematic review on the implementation of causal analysis in lower maturity organizations.

Table 1 Summary of the reviewed literature

Authors	Is it a Lower Maturity (L2 or L3) Organization?	Purpose of CAR analysis
Buglione, L et al. [7]	Yes	N/A just the challenge of implementing it at lower maturity organizations
Bhandari, I et al [8]	No	Process Improvement
Card D. N. [9]	No	Process Improvement
Card D. N.[10]	No	Project Defect Profile
Card D. N.[11]	N/A	Cost Saving by Defect Reduction
Mitzukami D.[12]	No	Project Defect Profile
Lezak M et al. [13]	No	Process Improvement
Fredericks M et al[14]	No	Process Improvement
Norman E.F. [15]	N/A	Defect Prediction
Bibi S. et al [16]	N/A	Project Defect Profile and Defect Prediction
Jacobs J. C. et al [17]	N/A	Project Defect Profile and Defect Prediction

In reference to our first question, we only found one reference [7] which shows that CAR can be implemented at a maturity level 3 organization (question **A**). The research done by Buglione and Abram [7], describes how it is possible to implement a CAR process area in organizations that have not yet achieved higher maturity rating. They base their implementation of the causal analysis meeting using Ishikawa (or Fishbones) diagrams [18] and defects have been classified using IBM's Orthogonal Defect Classification [4].

Finally, Table 1 shows that we have classified the results of Causal Analysis in three categories (question **B**). "Process Improvement" is the category that reports using Causal Analysis results as an input to process improvement initiatives. This use is aligned with [4] first proposal of ODC. The "Project Defect Profile" category represents initiatives that use defect data to compare the actual project to the historical projects database of the organization. They use Causal Analysis to understand deviations from the historical data and to implement corrective actions to their project and to the organization's process. In the "Defect Prediction", we have seen attempts to use artificial intelligence techniques to profile project's defects and to predict the number of remaining defects. Finally, "Cost Saving by Defect Reduction", means that the implementation of causal analysis can provide cost savings. All the results show the intended uses of causal analysis as they are recommended in the CMMI model. Unlike our research objective, none of the reviewed authors have linked causal analysis to training needs.

5. Development and Validation of the classification scheme

For a successful implementation of causal analysis session, the organization must develop a classification scheme that will enable developers to consistently classify defects. This section starts by presenting an overview of IBM's ODC taxonomy, which has served as the groundwork for our classification scheme. We then describe our implementation and our results.

5.1. Reference classification scheme: Orthogonal Defect Classification

At IBM [4, 20, 21] Defect Data classification has been used to drive Software Process Improvement initiatives. They call their process Orthogonal Defect Classification (ODC). Within the ODC context, developers classify defect data in orthogonal classifications. IBM proposes a taxonomy for defect classification based on the source of the defects:

- *Education*, in this category the developer did not understand some aspect of the product or the process. This

category is further divided into education in base code, education in new function, and other education, depending on what was not understood.

- *Communication*, in this category the developer did not receive the required information or the information was incorrect.
- *Oversight,* in this category the developer failed to consider all cases and conditions. Usually some detail of the problem or process was overlooked. The developer forgot something, had difficulty checking thoroughly, or did not have enough time to be thorough.
- *Transcript*. In this category the developer knew what to do and understood the item thoroughly but simply made a mistake. Transcription errors are typically caused by some problem in a procedure, for example, typing or copying a list of items.

In relation to ODC we have reviewed the work by [19], where they recognize the importance of having a classification scheme that is consistent with the organizations' tools and culture.

5.2. Classification Scheme implementation plan

The classification scheme was developed based on the advice in [19]. In their work, the authors recognize the benefits that the organizations can obtain if they take the time to develop their own classification scheme consistent with their information needs. As a result of this work, it was decided to develop a classification scheme that was suitable for our target organization. We divided the deployment plan in four major stages (as shown in Table 2). In the first stage, we developed the custom classification scheme. Secondly, the classification scheme was deployed to the organization through its defect tracking system (this is described in this section). Then, training in developing classification ability had to be implemented in order to achieve a reliable classification capability within the organization (described in the following section).

Table 2 Classification scheme deployment plan

Stage	Activity
Organizations Capability	Design and Validate classification scheme
Deploy Classification scheme	Modify defect tracking system
Training	Design and deploy classification training Evaluate training results

As we mentioned earlier, to minimize the rejection risk of the classification deployment, it is important that the classification taxonomy was consistent with its information needs and culture[19]. A custom classification was developed with the help of the practitioners. Furthermore, the researchers established equivalence between the organization's classification and ODC (see Table 3[1]).

The organization was already using a defect tracking system whose defect states were the ones proposed by the Microsoft Solution Framework for CMMI Process Improvement [20]. Therefore, implementing the ODClike classification went seamless in the developer's culture. The new classification scheme had to be implemented into the defect tracking system front end (a Microsoft InfoPath form template), which required only 4 hours work.

Table 3 Example of the organization's defect classification scheme

Defect Category	Classification Criteria	ODC Category Equivalence
Product Integration Error	Interface implementation does not match specification. Changes are not transferred to lower layers. Data not transferred from lower layers.	Communication
Error in use or configuration of user interface controls	Controls 'Freeze'. Paging not working in DataGrids. DataGrids missing headers.	Education
…	…	…

5.3. Description of the classification training and its results

For a successful implementation of the causal analysis meetings the developers must be consistent in their defect classification across the organization. A training event was prepared for the pilot project's developers.

Three of the organization's developers attended training. One of the trainees (C1) was a senior developer of the organization. The other two developers had experience in the pilot project's product line. Moreover, since we wanted to simulate the turnover rate of the organization, we included a fourth individual (O) to our experiment. This fourth subject had no contact with the organization and was given no training in the classification scheme.

The training consisted on a 4-hour seminar split into 2 days. The seminar was prepared and given by one of the

[1] Since defect data can be sensitive to the organization, we were asked to show only what was strictly necessary to communicate our results. Hence, we decided not to include the full classification scheme here.

researchers. This kind of training is the standard training effort that this organization invests on a given process. The organization's training events of this kind are similar to the ones described in [21]. During the seminar, an explanation of the causal analysis meetings purposes was given to the trainees. The classification scheme was presented with a sample defect for each one of the 12 classification categories.

For training evaluation, we provided the trainees with a set of 31 defects which we asked them to classify. We wanted a measure of how reliable the classification was between the trained developers. For this objective we applied Cohen's Kappa [22] between each of the subjects following a process similar to the one shown in [19, 23]. Cohen's Kappa is a statistical measure of inter-rater reliability, it is used to compare the level of agreement between two subjects who are classifying the same data set. We also applied the Kappa correlation to the outsider (O), and to an expert classifier (E). Our intention was to measure how the untrained individual rated against the other subjects. Furthermore, we applied a Fleiss' Kappa [24] to the trained group, and to the trained group plus the untrained subject. Fleiss' Kappa is a statistical measure for assessing the agreement of a number of subjects classifying a fixed data set. We expect that the examination of the results will show that trained individuals score higher correlation values than untrained individuals.

Table 4 shows the results for every pair: E represents the expert rater (one of the analysts who helped develop the classification). C1 – C3 represent the trained developers and O represents the outsider. While Table 5 presents the Fleiss' Kappa results with and without the Outsider.

Table 4 Cohen's Kappa between subjects

Subjects	Cohen's Kappa significance	Subjects	Cohen's Kappa significance
E – C1	0,73	C1 – C3	0,47
E – C2	0,55	C1 – O	0,36
E – C3	0,52	C2 – C3	0,52
E - O	0,34	C2 – O	0,37
C1 – C2	0,62	C3 – O	0,30

Table 5 Fleiss' Kappa calculation results

Subject Group	K agreement
E-C1-C2-C3	0,53
E-C1-C2-C3-O	0,44

Table 6 presents the significance agreement intervals for both Cohen's and Fleiss' Kappas proposed by [25]. Based on this table, we set the objective for accepting developers' classification agreement in the "Substantial Agreement" interval or above.

Table 6 Kappa significance table for Cohen and Fleiss

Cohen's Kappa	Significance
< 0	Poor Agreement
0.00 – 0,20	Slight Agreement
0,21 – 0,40	Fair Agreement
0,41 – 0,60	Moderate Agreement
0,61 – 0,80	Substantial Agreement
0,81 – 1	Almost Perfect Agreement

Table 5 shows that the group did not achieve the target score, which is an indication of the effectiveness of the training. Nevertheless, the most remarkable result is that the inclusion of an untrained individual does not reduce the overall agreement significance.

The result on the exercise provides enough confidence in that the developers (C1-C3) are able to consistently classify defects according to the organization's classification scheme. Such results will enable the training department to process the causal analysis meetings output as input for specific training to the projects. In addition to this, results also show the importance of training for supporting the classification scheme.

The results in Table 4 show that training has effect on the subjects' ability to classify. This conclusion is drawn from Table 4 where we can see that trained subjects score higher agreement among themselves than when there are compared with the untrained subject. On the other hand, some of the trained subjects have failed to meet the 0,61 expected rate. We interpret this as an indication that training can be improved. This conclusion is supported by the fact that all three trained subjects scored correlations in the same significance category (between 0,47 and 0,62). In any event, the results have shown that with a 4 hour training regular developers classify 50% better that junior developers. And senior developers achieve a classification ability that scores substantial agreement in the correlation table. Finally, the comparison of the results with the untrained subject proves that training has impact on the classification ability of the developers. Taking into consideration the inclusion of the untrained subject shown in Table 5, it seems that the organization would do better to improve the efficiency of the training given, rather than achieving 100% training coverage of their developers.

6. Conclusions and future work

In this article, we have proposed a process that links defect causal analysis to the training department. The process enables an organizational training department to show its contribution to the bottom line results of the organization. We have taken the first steps into implementing the process and we have also conducted a verification of these first steps.

It was determined that the key point for the success of the deployment of the process was the causal analysis session. An effort was made to reviewing the current state of the implementation of causal analysis in lower maturity organizations. The result was that we were able to find research that shows implementing causal analysis in lower maturity organizations is possible.

We developed a custom classification scheme for the organization, and cost-effectively modified the defect tracking system to deploy it.

Training in that classification scheme was given to a pilot project. The results show that training improves the developers' ability to classify defects.

In conclusion, causal analysis meetings have been successfully implemented in a maturity level 3 organization. Previous experiences encourage us to affirm that an implementation of the Causal Analysis and Resolution process area can be achieved at a lower maturity organization. We are now developing the following steps to achieve a full implementation of the proposed process.

7. References

1 Salas, E., and Cannon-Bowers, J.A.: 'THE SCIENCE OF TRAINING: A Decade of Progress', Annual Review of Psychology, 2001, 52, pp. 471-499

2 Chrissis, M.B., Konrad, M., and Shrum, S.: 'CMMI : guidelines for process integration and product improvement' Addison-Wesley, 2007.

3 Kirkpatrick, D.L., and Kirkpatrick, J.D.: 'Evaluating Training Programs: The Four Levels' Berrett-Koehler Publishers, 2006.

4 Chillarege, R.: 'Orthogonal Defect Classification-A Concept for In-Process Measurements', 1992

5 Phillips, J.J.: 'Return on Investment in Training and Performance Improvement Programs, Second Edition (Improving Human Performance)' Butterworth-Heinemann, 2003.

6 Kitchenham, B.: 'Procedures for Performing Systematic Reviews', Technical Report TR/SE-0401-ISSN, Keele University, UK, 2004, pp. 1353-7776

7 Buglione, L., and Abran, A.: 'Introducing Root-Cause Analysis and Orthogonal Defect Classification at Lower CMMI Maturity Levels ', 2006

8 Bhandari, I., Halliday, M.J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J.S., Lepori-Costello, C., Jasper, P.Y., Tarver, E.D., Lewis, C.C., and Yonezawa, M.: 'In-process improvement through defect data interpretation', IBM Syst. J, 1994, 33, (1), pp. 182-214

9 Card, D.N.: 'Defect-causal analysis drives down error rates', 1993, 10, (4), pp. 98-99

10 Card, D.N.: 'Managing Software Quality with Defects'. Proc. Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment, Year pp. 472-474

11 Card, D.N.: 'Learning from Our Mistakes with Defect Causal Analysis', 1998

12 Mizukami, D.: 'Analyzing Defects Can Tell a LOT About a Company'. Proc. SEPG Conference 2007, March 26 - 29, 2007 2007 pp. Pages

13 Leszak, M., Perry, D.E., and Stoll, D.: 'A case study in root cause defect analysis', Proceedings of the 22nd international conference on Software engineering, 2000, pp. 428-437

14 Fredericks, M., and Basili, V.: 'Using Defect Tracking and Analysis to Improve Software Quality ', Crosstalk, 1999

15 Norman, E.F.: 'A Critique of Software Defect Prediction Models', IEEE Transactions on Software Engineering, 1999

16 Bibi, S., Tsoumakas, G., Stamelos, I., and Vlahvas, I.: 'Software Defect Prediction Using Regression via Classification', Computer Systems and Applications, 2006. IEEE International Conference on., 2006, pp. 330-336

17 Jacobs, J.C., van Moll, J.H., Krause, P.J., Kusters, R.J., Trienekens, J.J.M., and Semiconductors, P.: 'Effects of Virtual Development on Product Quality: Exploring Defect Causes', Software Technology and Engineering Practice, 2003. Eleventh Annual International Workshop on, 2003, pp. 6-15

18 Ishikawa, K.: 'Guide to Quality Control' Asian Productivity Organization, 1986.

19 Freimut, B., Denger, C., and Ketterer, M.: 'An industrial case study of implementing and validating defect classification for process improvement and quality management', Software Metrics, 2005. 11th IEEE International Symposium, 2005, pp. 10 pp. %@ 1530-1435

20 Microsoft Solution Framework for CMMI Process Guideance Available at http://msdn2.microsoft.com/en-us/teamsystem/aa718802.aspx

21 Smith, C.: 'Achieving organizational training objectives with short course development', 13th Conference on Software Engineering Education & Training, 2000. Proceedings. , 2000, pp. 32-38

22 Cohen, J.: 'A Coefficient of Agreement for Nominal Scales', Educational and Psychological Measurement, 1960, 20, (1), pp. 37

23 Henningsson, K., and Wohlin, C.: 'Assuring fault classification agreement-an empirical evaluation', Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on, 2004, pp. 95-104

24 Fleiss, J.L.: 'Measuring nominal scale agreement among many raters', Psychological Bulletin, 1971, 76, (5), pp. 378-382

25 Landis, J.R., and Koch, G.G.: 'The Measurement of Observer Agreement for Categorical Data', 1977, 33, (1), pp. 159-174

Autonomous Reconfiguration Procedures for EJB-based Enterprise Applications

Thomas Vogel, Jens Bruhn, and Guido Wirtz
Distributed and Mobile Systems Group, University of Bamberg
Feldkirchenstraße 21, 96052 Bamberg, Germany
th.vogel@gmail.com, {jens.bruhn|guido.wirtz}@uni-bamberg.de

Abstract

Enterprise Applications (EA) are complex software systems for supporting the business of companies. Evolution of an EA should not affect its availability, e.g., because of a temporal shutdown, business operations may be affected. One possibility to address this problem is the seamless reconfiguration of the affected EA, i.e., applying the relevant changes while the system is running. Our approach to seamless reconfiguration focuses on component-oriented EAs. It is based on the Autonomic Computing infrastructure mKernel *that enables the management of EAs that are realized using Enterprise Java Beans (EJB) 3.0 technology. In contrast to other approaches that provide no or only limited reconfiguration facilities, our approach consists of a comprehensive set of steps, that perform fine-grained reconfiguration tasks. These steps can be combined into generic and autonomous reconfiguration procedures for EJB-based EAs. The procedures are not limited to a certain reconfiguration strategy. Instead, our approach provides several reusable strategies and is extensible w.r.t. the opportunity to integrate new ones.*

Keywords: maintenance, seamless reconfiguration, EJB

1. Introduction

Enterprise Applications (EA) are complex software systems for supporting the business of a company. According to Lehman's laws [10] software implementing real world applications like EAs must continually evolve, else their use and value would decline. The need for system evolution originates, e.g., from failures, inefficiencies or changes of the business or of the system environment that lead to new or changing requirements for EAs. Thus, system evolution can be categorized as *corrective* (removing software faults), *adaptive* (adjusting the system to the changing environment), or *perfective* (enhancing or improving the functional and non-functional system characteristics) (cf. [15, 18]). Due to the critical role of an EA within a company this evolution should not affect the availability of an EA and therefore the business operations. Otherwise, the company might miss business opportunities and loose reputation and trust. One approach to address this problem is the *seamless* reconfiguration, i.e., applying the relevant changes to the system while it is running. Except of delays in the response time reconfiguration should be transparent to the clients of the EA. This *post-deployment runtime evolution* can be seen as one critical challenge in software evolution [14]. To cope with this issue, the modularity of software systems, as proposed by the concept of *Component Orientation* (CO) [19], and the automation of system maintenance tasks, as described by the vision of *Autonomic Computing* (AC) [4, 8], can help. With the *mKernel* system [1, 2] a generic AC infrastructure is available that enables comprehensive management of component-oriented EAs that are realized with the *Enterprise Java Beans (EJB) 3.0* technology [3]. Based on *mKernel*, we provide a comprehensive set of steps, that are customizable and perform fine-grained reconfiguration tasks. These steps can be combined flexibly to generic and autonomous reconfiguration procedures for EJB-based EAs. Each of these procedures realizes a certain *reconfiguration strategy*, i.e., a certain way to perform a reconfiguration. Currently, our approach provides four reusable strategies that serve as templates for easing the planning and execution of a concrete reconfiguration.

The rest of the paper is structured as follows: Section 2 provides an overview to the background, namely system reconfiguration, CO and the AC infrastructure *mKernel*. Section 3 discusses related work, while section 4 presents our approach of reconfiguration procedures. Finally, the last section gives a conclusion and an outlook on future work.

2. Background

After introducing the basics of system reconfiguration, the concept of CO and relevant aspects of the EJB standard are presented. Finally, we describe how *mKernel* combines EJB with the vision of AC.

2.1. System Reconfiguration

The architecture of a software system is the *high-level organization of [its constituent] computational elements and the interactions between those elements* ([5], p.269). In this context, according to [13], there are two general approaches for software reconfiguration: *parameter adaptation* and *compositional adaptation*. The first one modifies variables of one or more elements that determine their behavior. The second one addresses structural reconfiguration through addition and removal of elements, including the manipulation of connections among them (cf. e.g. [9, 15, 17]). The weakness of parameter adaptation is that it allows only changes that were anticipated during development, because the elements have to provide the variables and react appropriately to their modifications. In contrast, compositional adaptation is intended for the dynamic and unanticipated reconfiguration of a system.

For carrying out a reconfiguration, two objectives are to be considered and desirable [9]. First, the reconfiguration should minimize the disruption to the system, i.e., the affected part of the system may notice delays but no failures, while the rest of the system should be able to continue its execution normally. Thus, reconfiguration should be carried out *seamlessly*. Second, a consistent state of the system must be preserved during and after reconfiguration. Consequently, a reconfiguration, like, e.g., the replacement of an element, may require to place the affected part of the system in a consistent state before structural changes are performed. A state is consistent if the affected elements are *quiescent* [9], i.e., none of them is currently engaged in servicing a request and none of them will initiate a request. Furthermore, no requests initiated by non-affected elements are forwarded to affected ones. To reach a quiescent state, requests that are currently serviced must be finished. New requests must be blocked except those which are needed to finish servicing ones. Otherwise, some elements are not able to reach a quiescent state and end up in a deadlock. Quiescence of the affected part of the system gives new elements the opportunity to be initialized in a state which is consistent with the rest of the system, and elements to be removed the opportunity to leave the system in a consistent state [9]. In case of an element replacement, this may include the need for transferring the internal state of a replaced element to a replacing one [15, 16].

How to apply changes are questions of reconfiguration strategies. In [16] the three strategies *Flash*, *Non-Interrupt* and *Interrupt* are presented. The *Flash* strategy reconfigures one element without concerning about other elements. Reconfiguration takes place immediately without handling existing interactions and the states of the affected elements. No state transfer is performed and existing connections to old elements are not updated. Therefore, these connections become invalid and are likely to cause errors. Finally, the system may become inconsistent. Consequently, *Flash* does not always perform a seamless and consistent reconfiguration. Nevertheless, it can be used, amongst others, for parameter adaptation or for reconfiguring elements not being critical for the consistency of the application. In contrast, the other strategies preserve consistency of the system and perform a seamless reconfiguration. The *Non-Interrupt* strategy supports the exchange of elements without the need for quiescence, hence reducing system disruption significantly. Both elements, the old one that is going to be replaced and the replacing one, are active. An intercepting facility forwards requests of already existing sessions to the old one and requests of new sessions to the replacing one. After all sessions on the old element have finished, it can be removed and only the new element is used. This strategy does not require a state transfer. It requires that the two elements can be used concurrently. The *Interrupt* strategy transfers the affected part of the system into a quiescent state before reconfiguration takes place. The states of the affected elements and existing connections between elements are handled, such that, e.g., an element replacement can be performed without causing any failures. After reconfiguration, the affected part of the system is released at once from the quiescent state, such that it can be assured that all elements and connections are reconfigured appropriately, before resuming their execution. Comparably with the other strategies, an advantage of requiring quiescence is that, e.g., an underlying database is not used during quiescence, which enables its consistent modification or transfer. Consideration of strategies is important to find the best way to reconfigure a system.

2.2. Component Orientation

The concept of *Component Orientation* (CO) [19] is a paradigm for the development of software systems in a modular way through functional decomposition. Such systems are composed of modules, called *Components*. A component encapsulates a certain functionality and provides it through contractually specified *Interfaces*. A component may use services from other components through their provided interfaces. An interface required by a component is called *Receptacle*. Consequently, a component-based system can be seen as a collection of loosely-coupled modules which collaborate among each other through their interfaces. Furthermore, a component can be deployed independently and is subject to composition by third parties [19]. Thus, CO addresses the complexity during development and deployment by modularity of requirements, architectures, designs, implementations and deployments. This modularity supports the partial reconfiguration of component oriented systems.

The *Enterprise Java Beans* standard (EJB), version 3.0, [3] is a component standard for the realization of component-oriented EAs on top of the *Java* programming language. It defines a sound component model that is based on so called *Enterprise Beans*, or *Beans* for short. There are two types of beans considered in the standard, namely *Message Driven Bean* and *Session Bean*. The former one is intended to be accessed through asynchronous message passing, and the latter one provides interfaces to access its encapsulated functionality. Session beans can be either *stateless* or *stateful*. An instance of a stateful session bean is exclusively used by a single client and retains its client-specific *Conversational State* across multiple invocations. In contrast, an instance of a stateless session bean is not exclusively used by a client. Moreover, each invocation from a client on the same reference may be executed on different instances. Thus, all instances of one stateless session bean are equivalent, and their states are client independent. Receptacles can be declared for session and message driven beans through *EJB References*. These can be connected to interfaces provided by session beans. Beans may be customized through *Simple Environment Entries* which can be interpreted as a kind of property. Before deploying beans, they must be configured, i.e., their properties must be set and their corresponding receptacles and interfaces must be connected. As unit of deployment the EJB standard defines the *EJB module* that must contain at least one bean. In the EJB context, parameter adaptation is performed through setting bean properties, and compositional adapta-

tion through (un)deploying modules and manipulating connections between beans. However, after the deployment of a module into an *EJB Container*, the runtime environment for components, the configurations of beans can not be changed.

2.3. Autonomic Computing and mKernel

The vision of *Autonomic Computing* (AC) [4, 8] addresses the management of systems at runtime. Its basic idea is to assign low level, administrative tasks to the managed system itself to disburden human administrators. The system manages itself according to the goals specified by the administrator. Autonomous management covers the four aspects *self-healing*, *self-protection*, *self-optimization*, and *self-configuration*. The last aspect addresses reconfiguration explicitly. Furthermore, each aspect can be mapped to at least one of the different kinds of system evolution discussed in section 1, namely corrective, adaptive, and perfective.

The *mKernel* system [1, 2] provides a generic AC infrastructure for EJB-based autonomous applications. It includes a comprehensive *Application Programming Interface* (API) of *sensors* and *effectors* through which the managed application can be inspected and manipulated by a higher level facility. Through this API, *mKernel* provides a reflective view, the *meta level*, of the managed application, the *base level*. Both levels are causally connected [11]. This reflective view enables the management of the application at three different levels of abstraction. The *Type Level* addresses information regarding types of the constituting elements of the managed application, i.e., artifacts being the result of development. The *Deployment Level* concentrates on a concrete configuration of the managed application that is deployed into a container. Finally, the *Instance Level* addresses the bean instances and interactions among them. With this multi-level view, subtle management operations are possible. As discussed in section 2.2, the EJB standard limits the configuration of bean properties and connections among beans to the deployment time, but *mKernel* enables the modifications of them at runtime. Together with supporting the lifecycle of EJB modules, *mKernel* provides runtime support for parameter and compositional adaptation. Nevertheless, the EJB specification is not violated or restricted by *mKernel*. Developers of EJB modules do not have to follow special guidelines beyond those of the EJB standard during development to enable the manageability of modules through *mKernel*. Thus, the developer can solely focus on the application logic while a preprocessing tool weaves the sensors and effectors into the EJB module. This approach maintains the idea of *Separation of Concerns*. Furthermore, *mKernel* is realized as a plugin for an existing EJB container and does not require any adjustments of the container implementation.

3. Related Work

Our approach to seamless reconfiguration is inspired by the work of Rosa, Rodrigues and Lopes [16] who present a framework for message-oriented systems that supports a fixed set of reconfiguration strategies. In contrast to their work, our approach is extensible w.r.t. the integration of new strategies. Moreover, the replacement of strategy elements is supported which provides additional flexibility. We support separation of concerns, because developers of EAs do not have to consider reconfigurations during development. Finally, the deployment and instance level are explicitly addressed, especially the transfer of conversational states of stateful session bean instances is supported. Our work addresses a different application area, namely EJB-based EAs.

Göbel and Nestler [6] extend the EJB specification by adding one more bean type, namely a composite bean. This composite encapsulates runtime adaptation by selecting different sub-components of the composite. Thus, the developer must consider this extension to the standard and only anticipated reconfiguration is possible that depends on the internals of the composite. Jarir, David, and Ledoux [7] enhance the EJB container to provide limited reconfiguration by intercepting calls to impose user-defined functionality. More possibilities are provided by Rutherford et al. [17], though their work is restricted to reconfiguration at the deployment level. They consider the management of the deployment lifecycle of modules and the modification of properties and of connections of beans. Nothing is said about the handling of bean instances, i.e., replacing bean instances together with their possible conversational states is not considered. In contrast, Matevska-Meyer, Olliges, and Hasselbring [12], who confine reconfiguration to redeploying modules, recognize the problem of the state transfer. They conclude that stateful beans are not safe to structural changes and provide no solution. Finally, the research group of the *Peking University Application Server* (PKUAS) [20] has implemented an own EJB container that incorporates the necessary technological facilities for updating modules including bean instances and state transfer. Thus, they consider the deployment and instance level. But they do not support higher-level facilities, like reconfiguration strategies that may simplify the role of administrators.

4. Autonomous Reconfiguration Procedures

Our approach to seamless reconfiguration of EJB-based EAs covers parameter and compositional adaptation. To meet various reconfiguration needs we identified and provide a comprehensive and complete set of customizable and reusable *steps*, that are described in table 1. The first column contains identifiers for the steps. The second column covers a short description of the particular step. Each step performs a special reconfiguration task, like, e.g., the deployment of a module (step a) or the establishment of connections between beans (step l). Steps are realized by so called *executors* that are based on the *mKernel* API. This is depicted on the left hand side in the reconfiguration model in figure 1. Our implementation provides default executors for all steps, except the step that is intended for reconfiguration of databases. Nevertheless, administrators have the freedom to provide *custom executors* for arbitrary

ID	Step	dep.	F	NI	I	I/NI
a	Deployment of the new EJB module. Setting the *Simple Environment Entries* and connecting the *EJB Reference* of its beans.	/	1	1	1	1
b	Declaration of the quiescence region which comprises those beans or whole modules that must be quiescent at a later point in time. For module replacement, this region is the module which is going to be replaced.	/	-	-	2	2
c	Start tracking and collecting session bean instances of beans of the quiescence region to get to know the instances that are handled with step f.	b	-	-	3	3
d	Initializing the quiescence, i.e., initializing the blocking of calls on the instances of beans of the quiescence region. The region becomes quiescent after finishing current calls.	b	-	-	4	6
e	Waiting until the quiescence region becomes quiescent.	d	-	-	5	7
f	Extracting the state of stateful session bean instances being collected because of step c.	c, e	-	-	6	8
g	Extracting the database that underlies the quiescence region.	e, f	-	-	7	-
h	Transfer or modify the database.	g	-	-	8	-
i	Starting of the new EJB module.	a	2	2	9	4
j	Modifying (optional) and injecting the states, being extracted at step f, to newly created instances of the corresponding stateful session beans of the new EJB module.	f, h, i	-	-	10	9
k	Publishing the references of the new bean instances that have been the target of the transfer of step j. Client components holding references to replaced stateful session bean instances are provided with the corresponding new reference to the replacing instance.	j	-	-	11	10
l	Re-route connections that are newly established. The source of the these connections are client components of the new/old EJB module and the target of these connections shifts from the beans of the old EJB module to the beans of the new EJB module.	h, i	3	3	12	5
m	Re-route already existing connections, i.e., clients of the old module holding references to bean instances of the old module are provided with new references to bean instances of the new module. In case of an I or I/NI this step only considers connections whose targets are bean instances which have not been transferred. Connections to transferred bean instances are already covered by step k. In case of NI this step is optional and only consistently applicable if the target of the connection is a stateless session bean instance.	h, i	-	4	13	11
n	Release the quiescence region, i.e., blocked calls and eventually blocked bean instance lookup requests are released and continue executing through using the reference already held before quiescence or the reference provided in steps k, l or m.	d, h, k, l, m	-	-	14	12
o	Stop and optionally undeploy the old EJB module if the old module is not used any more, or in case of the F strategy, should not be used any more through existing connections.	g, k, l, m	4	5	15	13

Table 1. Reconfiguration procedures and their steps

steps replacing the default ones. In this way, special requirements for reconfiguring concrete applications can be fulfilled.

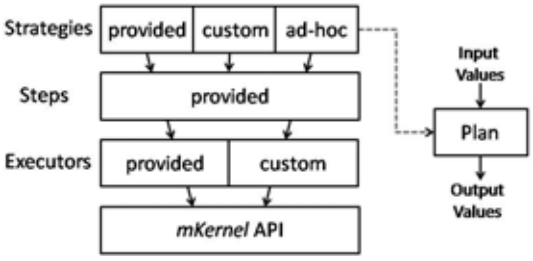


Figure 1. Reconfiguration Model

The provided steps are the basis for the *strategies* (see figure 1). Therefore, steps can be combined into generic and autonomous reconfiguration procedures. A procedure must fulfill the dependencies between its constituting steps. The third column of table 1 contains the steps each step depends on transitively. A '/' depicts that the particular step does not depend on any other step. Starting a new EJB module, e.g., requires that the module has been deployed before, therefore step i depends on a. Nevertheless, as some steps may be optional, corresponding dependencies need not to be met. For the case, that no state transfer is necessary, steps c, f, j and k can be omitted, and the step of stopping the old EJB module (o) does not depend on step k, but only on g, l and m. Consequently, these dependencies are influenced by a concrete arrangement of steps that may skip optional steps and by the concrete modules and beans each step is addressing. However, dependencies can be used to find basic restrictions in ordering the steps or potentials for parallel execution of steps. There exists, e.g., no dependency between the steps a and b, such that they can be executed in arbitrary order or even in parallel. The reusability of each step, the flexibility in ordering the steps and the possibility to omit optional steps enable various combinations of steps into generic reconfiguration procedures. Each procedure realizes a certain

reusable reconfiguration strategy. Therefore, administrators can develop *custom* strategies, that may be derived from others or that may be completely new ones. Even, a dynamic arrangement of steps during runtime is possible, resulting in *ad-hoc* strategies (see figure 1). Our approach provides the four strategies *Flash* (F), *Non-Interrupt* (NI), *Interrupt* (I), and *Interrupt/Non-Interrupt* (I/NI). Besides the first three ones, already presented in section 2.1, we identified I/NI as an additional new strategy for replacing modules. It is a mixture of the strategies I and NI. Its idea is that the new and old module are running concurrently, but newly created sessions are forwarded to the new module and start processing immediately. Already running sessions on the old module will not run until they finish, like it is done with the NI strategy. In contrast, they are driven into a quiescent state and their instances of the stateful session beans are transferred to the new module, where finally the sessions continue their processing. The advantage of I/NI is that system disruption is minimized because newly created sessions are not blocked from servicing requests. Therefore, the underlying database must be usable by both modules concurrently. Furthermore, the old module is removed from the system consistently.

In the following, we discuss in detail how the four strategies F, NI, I and I/NI can be applied for the case of replacing an EJB module with an alternative implementation. Therefore, each of the last four columns of table 1 describes a realizing procedure for the corresponding strategy. The entries of these columns are to be read as follows. A step that is not applicable or available within a strategy is denoted with a '-'. Otherwise, the number indicates the position of this step within the procedure.

For the F strategy only the deployment level is relevant since existing bean instances and connections among them are not handled. The other strategies distinguish between already existing connections and newly established connections of bean instances, hence considering the instance level of the application. Re-routing a connection before it is created is always possible. Re-routing existing connections is feasible if the target of the connection is a stateless session bean, because the states of stateless session bean instances are client independent (see section 2.2) and both beans, replaced and replacing one, provide the same functionality. However, if the target is a stateful session bean, an existing connection can only be modified consistently if the conversational state is transferred to the corresponding target instance, otherwise the client-specific state would get lost. As described in section 2.1 a state transfer requires quiescence of the affected beans, i.e., *all* instances of the affected beans must be quiescent. Reaching quiescence is simplified by the EJB standard because bean instances are per definition non-reentrant and are not allowed to perform any kind of thread handling. Quiescence is performed by the steps b, d, e and n. Another motivation for quiescence is the need to transfer or modify the database (steps g and h) that underlies the modules, i.e., the old and the new module must be either quiescent or in a stopped state. This is addressed by the I strategy. In summary, a state transfer (steps c, f, j and k) is only required if stateful session beans are involved and an I or I/NI strategy should be used. Without state transfer and database reconfiguration there is no need for quiescence, hence F or NI are the preferable strategies.

For each step being part of a concrete strategy an executor must be assigned. A step executor may define input and output parameters. Inputs represent information required for an appropriate execution and information about execution results are provided through outputs. Outputs can be mapped to inputs of subsequent executors. E.g., our executor implementation for step f outputs the extracted conversational states which are used as inputs for the executor of step j. At strategy level, inputs can also be specified. These can be connected to those executors inputs for which no matching outputs are given. Likewise, outputs can be defined for a strategy that provide information about execution results of an instantiated strategy to an administrator. Therefore, executor outputs can be connected to strategy outputs. To sum up, a concrete strategy consists of a set of steps together with their executors, specifications of inputs and outputs at the strategy level, and mapping specifications between parameters. In addition to the dependencies described in the third column of table 1, these mappings may introduce additional dependencies. A strategy is *valid* if there are no circular dependencies and if all executor inputs are connected either to strategy inputs or outputs of preceding executors. As long as the dependencies are fulfilled, the order of steps may change within a procedure. Thus, it is conceivable that a strategy is realized by several procedures, i.e., different orders of steps. The procedures described in the last four columns of table 1 reflect the provided implementations. For a concrete reconfiguration need, an administrator must provide a reconfiguration plan, i.e., a strategy must be chosen, instantiated and configured. Consequently, the plan consists only of the selected strategy and of values for strategy inputs (see right hand side of the figure 1). During execution, parameter values are injected to the relevant step executors. Therefore, the reconfiguration can be executed without further interaction need. Thus, an administrator only needs to know *what* a strategy is doing, but not *how* it is realized.

Our current implementation supports all four aforementioned strategies to replace one EJB module with an alternative implementation of this module. The reconfiguration plan for each strategy requires only the identifiers of the replaced module and of the replacing module type as input values to perform the reconfiguration autonomically. Nevertheless, the following restrictions must be fulfilled.

1. The replacing module must provide implementations for at least those interfaces that are provided by the replaced module and referenced by clients. This implies that the replacing module must fulfill the same contracts specified by these interfaces as the replaced module.
2. Each interface identified through restriction 1 must be implemented by exactly one session bean inside both, replaced and replacing modules.
3. For all required *EJB References* of each of the session beans providing at least one of the interfaces identi-

fied through restriction 1, there must exist appropriate providers. An appropriate provider is a session bean which is not part of the replaced module. If the provider is part of the replacing module, this restriction must hold recursively. All *EJB References* of providers not being part of the replacing module must be connected to interfaces, recursively.

4. For each bean of the replaced module, there exists one bean in the replacing module that provides at least the same interfaces w.r.t. restriction 1.
5. For stateful session beans, the state transfer at instance level is only performed for those fields of the replaced bean - regardless of their access modifiers - for which there exists a matching counterpart in the replacing bean. In this context, two fields are matching if they have the same name and type in both, the replacing and the replaced beans.

Though these restrictions are imposed on modules, the alternative implementation of the replacing module may eliminate failures in the behavior of the replaced one or it may be a more efficient implementation. Additionally, the integration of new functionality through adopting new or enhanced interfaces by the replacing module is possible.

5. Conclusion and Future Work

With this paper we presented a flexible approach to seamless reconfiguration of EJB-based EA that need not to be anticipated during EA development, hence it maintains the idea of separation of concerns. By providing generic and reusable procedures an administrator is freed from handling fine-grained reconfiguration tasks for each reconfiguration need. Instead of prescribing how a reconfiguration should be applied, the administrator can choose between several strategies. Thus, the role of the administrator is reduced to selecting an appropriate strategy and creating a reconfiguration plan that configures a generic procedure for a concrete reconfiguration need. The reconfiguration is performed autonomically.

As future work, it would be desirable if a mixture of the presented strategies could be applied for the replacement of a module, i.e., a strategy is applied only to a subset of beans of the module instead to all of its beans. Thus, disjoint subsets of beans can be reconfigured individually. Perhaps, this can be even broken down to the instance level. Finally, first considerations are made to weaken the restrictions of our current executors, e.g., to enable the replacement of n modules with m modules. Additionally, we investigate x-to-y relations for the bean replacement instead of only x-to-1 relations.

References

[1] J. Bruhn, C. Niklaus, T. Vogel, and G. Wirtz. Comprehensive support for management of Enterprise Applications. In *Proceedings of the 6th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2008)*, pages 755–762, Doha, Katar, March 2008. IEEE.

[2] J. Bruhn and G. Wirtz. mKernel: A Manageable Kernel for EJB-based Systems. In *1st ICST/ACM International Conference on Autonomic Computing and Communication Systems (Autonomics 2007)*, Rome, Italy, October 2007. ACM.

[3] L. DeMichiel and M. Keith. JSR 220: Enterprise JavaBeans, Version 3.0: EJB Core Contracts and Requirements. 2006.

[4] A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.

[5] D. Garlan and D. E. Perry. Introduction to the special issue on software architecture. *IEEE Transactions on Software Engineering*, 21(4):269–274, 1995.

[6] S. Göbel and M. Nestler. Composite Component Support for EJB. In *WISICT '04: Proceedings of the Winter International Synposium on Information and Communication Technologies*, pages 1–6. Trinity College Dublin, 2004.

[7] Z. Jarir, P.-C. David, and T. Ledoux. Dynamic Adaptability of Services in Enterprise JavaBeans Architecture. In *Seventh International Workshop on Component-Oriented Programming (WCOP'02) at ECOOP*, 2002.

[8] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.

[9] J. Kramer and J. Magee. The evolving philosophers problem: dynamic change management. *Transactions on Software Engineering*, 16(11):1293–1306, Nov 1990.

[10] M. Lehman, J. Ramil, P. Wernick, D. Perry, and W. Turski. Metrics and Laws of Software Evolution-The Nineties View. *Proceedings of the Fourth International Software Metrics Symposium*, pages 20–32, 1997.

[11] P. Maes. Concepts and experiments in computational reflection. In *OOPSLA '87: Conference proceedings on Object-oriented programming systems, languages and applications*, pages 147–155, New York, NY, USA, 1987. ACM.

[12] J. Matevska, S. Olliges, and W. Hasselbring. Runtime reconfiguration of J2EE applications. In *1st French Conference on Software Deployment and (Re)Configuration, DECOR04*, pages 77–84, 2004.

[13] P. McKinley, S. Sadjadi, E. Kasten, and B. Cheng. Composing Adaptive Software. *Computer*, 37(07):56–64, 2004.

[14] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri. Challenges in Software Evolution. *Eighth International Workshop on Principles of Software Evolution*, pages 13–22, 5-6 Sept. 2005.

[15] P. Oreizy, N. Medvidovic, and R. N. Taylor. Architecture-based Runtime Software Evolution. In *ICSE '98: Proceedings of the 20th international Conference on Software Engineering*, pages 177–186, Washington, DC, USA, 1998.

[16] L. Rosa, L. Rodrigues, and A. Lopes. A framework to support multiple reconfiguration strategies. In *Proceedings of the First International Conference on Autonomic Computing and Communication Systems (Autonomics 2007)*, 2007.

[17] M. J. Rutherford, K. M. Anderson, A. Carzaniga, D. Heimbigner, and A. L. Wolf. Reconfiguration in the Enterprise JavaBean Component Model. In *CD '02: Proceedings of the IFIP/ACM Working Conference on Component Deployment*, pages 67–81, London, UK, 2002. Springer-Verlag.

[18] E. B. Swanson. The Dimensions of Maintenance. In *ICSE '76: Proceedings of the 2nd International Conference on Software Engineering*, pages 492–497, Los Alamitos, CA, USA, 1976.

[19] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2002.

[20] Q. Wang, F. Chen, H. Mei, and F. Yang. An Application Server to Support Online Evolution. In *International Conference on Software Maintenance*, pages 131–140, 2002.

Cross-Language Clone Detection

Nicholas A. Kraft, Brandon W. Bonds, and Randy K. Smith
Department of Computer Science
The University of Alabama
Tuscaloosa, AL 35487, USA
{nkraft,bbonds,rsmith}@cs.ua.edu

Abstract

Code duplication is a common software development practice that introduces several similar or identical segments of code, or code clones. In addition, there is currently a trend towards the use of multiple languages in the development of software systems. While there has been much work on clone detection and increased interest in studies of multi-language software systems, there have been no studies of code clones that span multiple languages, which we term cross-language code clones. In this paper we describe an approach for cross-language clone detection. We then introduce a tool, which is based on the CodeDOM library that is included with the Microsoft .NET Framework, to demonstrate the existence of cross-language clones in a real software system that contains both C# and Visual Basic.NET source code. Because our clone detection algorithm operates on a tree structure, other tree-based clone detection algorithms could be substituted in the implementation of our tool.

1. Introduction

Code duplication is a common software development practice that introduces several similar or identical segments of code, or *code clones*, and has many forms: language construct recurrence, pattern or paradigm adherence, framework reuse, and copy-paste replication. There are some advantages to duplicating code; developer productivity can be improved through the use of copy-paste to quickly replicate functionality. However, there are also important disadvantages to duplicating code; program comprehensibility, maintainability, and correctness can all be adversely affected by code duplication practices, particularly copy-paste replication. For example, when a change is to be made to a duplicated piece of code, a developer must determine whether the change should be made to all duplicate instances (clones) of that code. If the developer makes an incorrect determination, or accidentally misses a duplicate instance, then a bug is introduced.

Because code clones can have a negative impact on system quality, techniques and tools to eliminate code clones have been proposed. For example, techniques for automatic refactoring have been described [3, 5] and tools to guide manual refactoring [12] and to allow meta-level refactoring [4] have been developed. However, studies have shown that some code clones are inherent and that their elimination is not always desirable [15, 16]. Therefore, additional techniques, such as linked editing [38], are needed to mitigate the unneccessary maintenance costs associated with code clones [11].

Irrespective of the benefits or drawbacks attributed to a code clone or a category of code clones, techniques and tools to detect code clones are needed, and much work on clone detection has been performed. These techniques and tools use as inputs a variety of program representations, including source code [1, 2, 8, 14, 25], parse or abstract syntax trees [5, 9, 13, 19, 20, 37, 39], abstract semantic graphs [28], and program dependence graphs [17, 22]. Furthermore, these techniques and tools use a variety of matching approaches, including string or token comparison [1, 2, 8, 14], metric comparison [28], hashing [5], subgraph isomorphism [17, 22], feature vectors [19, 13], frequent item sets [25, 39], suffix trees [20, 37], and structural abstraction [9]. Direct comparisons and evaluations of some of these techniques and tools are provided in the literature [6, 33].

While several of the aforementioned clone detection techniques can be adapted to multiple languages [32] and some of the aforementioned clone detection tools accept more than one source language (for example, CCFinder [14] can detect code clones in Java, C, C++, C#, Visual Basic, and COBOL programs), all of the techniques and tools focus on detecting same-language code clones, that is, code clones for which each associated code segment is written in the same source language. However, there is currently a trend towards the development of heterogeneous soft-

ware systems in which multiple languages are used. The Microsoft .NET Framework [29], which leverages a common byte code format and virtual machine to allow programs written in different languages to interact, is a central actor in this trend. While this trend has caused increased interest in studying multi-language software systems [18, 27, 26, 30, 35], there are currently no approaches for detecting code clones that span multiple languages or studies of such clones.

In this paper we describe an approach for detecting code clones that span multiple languages, which we term *cross-language code clones*. Our approach is complementary to other clone detection techniques, as we are focused primarily on issues associated with detecting cross-language code clones and not on the mechanics of clone detection. We also introduce a tool, C2D2, that implements cross-language clone detection for the .NET Framework. We then apply C2D2 to two subsystems of a real, open-source software system, Mono™ [31]. Our results demonstrate the existence of cross-language code clones, specifically code clones that span the C# and Visual Basic.NET languages.

The rest of the paper is organized as follows. In Section 2 we provide background information about the libraries that we use to implement our cross-language clone detection system, C2D2, and in Section 3 we provide an overview of C2D2. In Section 4 we describe a case study and report our results. Finally, we discuss related work in Section 5 and conclude in Section 6.

2. Background

In this section we provide background information about the Code Document Object Model (CodeDOM) and NRefactory. Both are libraries that we use to implement our cross-language clone detection system, C2D2.

2.1. CodeDOM

The Microsoft .NET Framework includes the Code Document Object Model (CodeDOM) library, which provides a language-independent metamodel for representing source code [7]. CodeDOM exists primarily to allow developers of programs that emit source code, such as the Visual Studio.NET forms designer, to emit source code in multiple programming languages from a common representation. A CodeDOM graph is a graph of CodeDOM nodes that represent the logical structure of source code and can be used both to generate source code in any language for which an implementation of the `ICodeGenerator` interface is provided.

CodeDOM is similar to other metamodels for abstract syntax trees [10, 21] but is more similar to the Dagstuhl Middle Metamodel (DMM) [23] and the Common Meta-Model (CMM) [36]. Like the DMM and the CMM, CodeDOM provides classes that represent many common object-oriented language constructs, including namespaces, type declarations (classes, structs, and enumerations), methods, fields, exceptions, and control statements. Also like the CMM, CodeDOM provides classes that represent statements and expressions, and is extensible. The expressiveness of CodeDOM is limited by the necessity of providing the option to generate code in any language in the .NET-family. Thus, CodeDOM can express only constructs that are available in all .NET languages.

Language constructs that are not directly supported by CodeDOM can be represented using a generic node, or "snippet" node. However, in many cases, source code can be restructured to avoid the use of snippets. For example, the switch statement is not present in all .NET languages, but can be restructured as a series of `if` statements. Such a restructuring ensures that the desired functionality can be achieved across all .NET languages.

2.2. NRefactory

Microsoft implements the interfaces in the CodeDOM library that allow a developer to populate a CodeDOM graph programmatically, but they do not implement the interfaces that would allow a developer to populate a CodeDOM graph from existing source code. The SharpDevelop IDE [34] includes the NRefactory library, which provides parsers for both C# and Visual Basic.NET; each of these parsers builds an internal abstract syntax tree (AST) representation of the input program. The `CodeDomOutputVisitor` class of NRefactory traverses the internal AST to produce a CodeDOM graph that represents source code that is semantically equivalent to the source code provided as input to the parser. However, because the expressiveness of the CodeDOM is limited, the source code generated from the produced CodeDOM graph may not be syntatically identical to the original source code.

3. C2D2

In this section we present our cross-language clone detection system, C2D2, including an overview of the system, a description of changes we made to NRefactory, and the details of our clone detection algorithm.

Figure 1 illustrates an overview of the C2D2 system, which takes as input one or more C# or Visual Basic.NET source files and produces as output a listing of detected clones. The source files, which are shown in the upper left of the figure, are read by the convertor component, which is shown towards the lower left of the figure. The convertor component converts each source file to a CodeDOM graph

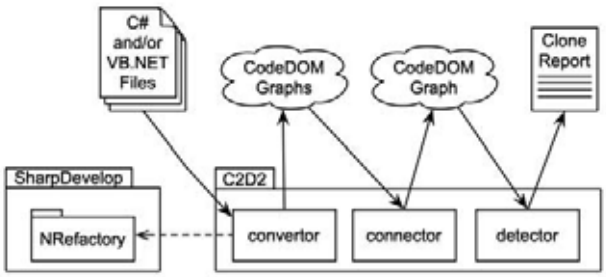


Figure 1. Overview of C2D2. *Dashed lines represent "use" dependencies. Solid lines represent data flow. Tabbed files are provided by the user. Clouds represent internal data structures. Lined files are generated by the system.*

by passing it to the NRefactory library, which is shown in the lower left of the figure. The connector component, which is shown in the lower middle of the figure, connects the collection of CodeDOM graphs produced by the convertor component to form a unified CodeDOM graph. The detector component, which is shown in the lower right of the figure, detects clones in the unified CodeDOM graph produced by the connector component. Output of the detector component is a clone report that lists the detected code clones.

3.1. Changes to NRefactory

We made several changes to the NRefactory library to fix bugs and to provide new functionality. With these changes, the CodeDOM graphs produced by the NRefactory parsers are usable for clone detection. Our corrections and additions to the NRefactory code are located primarily in the `CodeDomOutputVisitor` class. For example, in C# the `using` keyword can be followed by either an expression or a statement; however, only the expression case was handled by the version of NRefactory that we downloaded (version 2.2.0.2532). We added code to handle the statement case.

The parsers provided by NRefactory annotate some (but not all) AST nodes with line and column numbers, but do not propogate this information to the produced CodeDOM graphs. However, CodeDOM allows graph nodes to be annotated with a dictionary of user data via the `UserData` property. We leveraged this property to add line and column numbers (along with other auxilary data provided by the NRefactory lexers and parsers) to the nodes in the CodeDOM graph.

Our efforts to correct and enhance the NRefactory library are ongoing. Despite our efforts, there remain C# constructs that are either not properly parsed or not properly translated to CodeDOM; we are not yet able to obtain a CodeDOM graph for NRefactory due to these deficiencies. Nonetheless, our initial efforts allow the CodeDOM graphs produced by our modified version of NRefactory to be used for our feasibility study.

3.2. Clone Detection Algorithm

We utilize a hybrid token/tree-based algorithm for clone detection. Tree-based detection algorithms tend to be slower and more complex than token-based algorithms, so we wished to use a token-based algorithm to complete our feasibility study of cross-language clone detection. However, our input structure is a tree, not a token stream; thus, we created a hybrid token/tree-based algorithm.

The first step in our algorithm is to traverse the tree that underlies the unified CodeDOM graph and to create and store a string token for each leaf node in the tree. We perform the traversal depth-first to allow each interior node in the tree to access the tokens of its children during the traversal. An interior node stores the tokens of its children in a list and in prefix order. After traversal of the tree is complete, the list in the root node contains the token for each node in the tree, and so on down the tree. During the traversal, we also store each tree node in a list of nodes of the same type; one list exists for each CodeDOM node type.

Our matching algorithm is based on the Levenshtein distance algorithm [24]. The Levenshtein distance between two strings is the minimum number of character insertions, deletions, or substitutions required to transform one of the strings into the other string. We adapt this algorithm to work on lists of tokens representing CodeDOM nodes (recall that each CodeDOM node in the tree, except for a leaf node, contains the list of tokens for its children), which in turn represent code segments. We determine the minimum number of token insertions, deletions, or substitutions required to transform one list of tokens into the other list of tokens. We store our results in a data structure that stores the number of tokens cloned and the percentage matching (cloning) between two lists of tokens (code segments).

We only apply our matching algorithm on CodeDOM nodes of the same type. This is equivalent to only applying the algorithm on AST nodes of the same type, or to only attempting to match code segments of the same syntactic form. By intelligently applying our matching algorithm we reduce the complexity of the clone detection process and greatly improve performance. In addition, we eliminate the possibility of detecting a clone that crosses a block boundary; thus, we retain a key advantage that tree-based clone detection approaches have over token-based approaches.

Our matching algorithm takes four parameters. The first parameter specifies a minimum number of tokens that a list may contain to be considered, and the second parameter

		Percentage Matching					Tokens Matched							
	Total	30–40	40–50	50–60	60–70	70–100	0–10	10–20	20–30	30–40	40–50	50–60	60–70	70+
# of Clones	8,029	7,347	361	188	10	0	0	805	4,328	2,590	278	25	3	0

Table 1. Cross-Language Clones. *The number of cross-language clones detected by C2D2, categorized by percentage matching and by tokens matched.*

```
static public void EmitInt (ILGenerator ig, int i)
{
   switch (i){
   case -1:
      ig.Emit (OpCodes.Ldc_I4_M1);
      break;
   ...
   default:
      if (i >= -128 && i <= 127){
         ig.Emit (OpCodes.Ldc_I4_S, (sbyte) i);
      } else
         ig.Emit (OpCodes.Ldc_I4, i);
      break;
   }
}
```

```
Shared Sub EmitLoadI4Value(ByVal Info As EmitInfo, \
                           ByVal I As Integer, \
                           ByVal TypeToPushOnStack As Type)
   TypeToPushOnStack = \
      Helper.GetTypeOrTypeBuilder(TypeToPushOnStack)
   Select Case I
      Case -1
         Info.ILGen.Emit(OpCodes.Ldc_I4_M1)
      ...
      Case SByte.MinValue To SByte.MaxValue
         Dim sbit As SByte = CSByte(I)
         Info.ILGen.Emit(OpCodes.Ldc_I4_S, sbit)
      Case Else
         Info.ILGen.Emit(OpCodes.Ldc_I4, I)
   End Select
   Info.Stack.Push(TypeToPushOnStack)
End Sub
```

Figure 2. Example Cross-Language Clone. *A portion of a cross-language clone detected by C2D2. The left side of the figure lists a C# code segment from* `mcs/constant.cs` *and the right side of the figure lists a Visual Basic.NET code segment from* `vbnc/source/Emit/Emitter.vb`*. The full clone spans lines 805–855 of* `mcs/constant.cs` *and lines 1384–1414 of* `vbnc/source/Emit/Emitter.vb`*, and has a percentage matching of 58 (69 of 118 tokens matched).*

specifies the corresponding maximum number. These parameters can be used to reduce the number of matches attempted by the algorithm; please note that these parameters specify the numbers of tokens a list may contain to be considered by the algorithm and not the numbers of tokens that a clone may contain. The third parameter specifies a minimum percentage matching. This parameter can be used to filter clones that do not reach the specified percentage of similarity. Finally, the fourth parameter specifies whether to attempt to detect same-language clones (in addition to cross-language clones).

4. Case Study

In this section we describe a case study used to evaluate the feasibility of our approach. We use the C# and Visual Basic.NET compilers from Mono [31], version 1.2.6, as the test case. We chose these compilers because they are open-source and are part of a large, multi-language software system. Furthermore, as compilers for a common back end (the .NET runtime environment) they are likely to have common functionality in their code generators.

The C# compiler, mcs, consists of 38 C# files that contain 49,216 lines of non-blank, uncommented, non-preprocessed source code. The Visual Basic.NET compiler, vbnc, consists of 422 Visual Basic.NET files that contain 52,161 lines of non-blank, uncommented, non-preprocessed source code. Thus, our test case totals 460 source files and 101,377 lines of source code. Before running C2D2 on the test case, we configured it as follows: minimum number of tokens (50), maximum number of tokens (200), minimum cloning percentage (30), and attempt to detect same-language clones (no). We performed all experiments on a workstation with an AMD Athlon™ 64 X2 3800+ processor and 1 GB RAM; the operating system is Microsoft Windows XP Professional SP2 32-bit. Using the above configuration and the test case as input, the total execution time for C2D2 is 21 minutes and 23 seconds (wall clock time).

Table 1 summarizes the results of the execution. A total of 8,029 cross-language clones were detected; however, 7,708 of the detected clones had a percentage matching of less than 50, leaving only 198 clones with a percentage matching of 50 or greater. Of those 198 clones, only 10 have a percentage matching of 60 or greater and none have a percentage matching of 70 or greater. At first glance, these results do not appear to demonstrate the existence of meaningful cross-language clones. Yet, the example clone illustrated

in Figure 2 shows that percentage matching is a potentially misleading metric for cross-language clones detected using our algorithm. Figure 2 illustrates a cross-language clone that consists of a segment of C# code and a segment of Visual Basic.NET code. If it were not for the stack manipulation performed in the Visual Basic version of the code then the functionality would be identical. This example clone is a *type 3* clone [6], i.e., a clone in which identifiers have been changed and statements have been changed, added, or removed (added in this case). However, the percentage matching for the clone is only 58% (69 of 118 tokens matched), which might seem low.

Despite the discovery of the cross-language clone illustrated in Figure 2 and others, when compared to the results of other clone studies, the number of detected clones with a significant percentage matching appears to be low for the amount of code included in the test case. There are several possible explanations for this. First, as noted above, percentage matching seems to be a misleading metric for evaluating cross-language clones detected using our algorithm. Second, Mono simply might not have many cross-language clones; this seems likely given that different projects within Mono often have disparate development teams. However, this would indicate that study of additional multi-language software systems is warranted. We plan to undertake such studies in the future. A third possibility is that C2D2, while capable of finding some cross-language clones (as demonstrated above), might miss other cross-language clones. This could be due to the clone detection algorithm employed. Again, this would indicate that further study, including applying more advanced tree-based clone detection techniques, is warranted.

5. Related Work

In this section we discuss related work, which we divide into two categories: (1) studies of clones and (2) studies of multi-language software systems.

5.1. Studies of Clones

Many techniques and tools for clone detection exist. These techniques and tools operate on a variety of program abstractions, including strings [1, 2, 8], tokens [14, 25], trees [5, 9, 13, 19, 20, 37, 39], and graphs [17, 22, 28]. Because our approach detects clones on a tree structure, other tree-based clone detection algorithms could be substituted for the one we present.

Studies investigating the presence of code clones in single-language software systems have found significant amounts of duplicated code within these systems. These studies, known as clone coverage studies, suggest that it is not uncommon to have over 20% cloned code in a software system. For example, CCFinder detected almost 30% cloned code in version 1.3.0 of the JDK [14], and CP-Miner detected over 22% cloned code in version 2.6.6 of the Linux kernel [25]. In addition, one study reported over 59% cloned code in a COBOL payroll application [8].

5.2. Cross-Language Studies of Multi-Language Software Systems

There is currently a trend towards the development of heterogeneous software systems in which multiple languages are used [18]. This trend has caused increased interest in studying multi-language software systems. Topics of interest include modeling, usability, tool support, as well as maintenance processes, which would include clone detection and evolution.

Linos, et al. [27] and Moise and Wong [30] present studies of cross-language dependencies found in software systems written in C/C++ and Java. Strein, et al. [35] present an approach to cross-language program analysis for refactoring and a prototype tool that handles both C# and Visual Basic.NET; however, they do not leverage CodeDOM for their implementation. Finally, Linos, et al. [26] present an approach to computation of metrics on MSIL (Microsoft Intermediate Language). Their ultimate goal is to determine whether computation of metrics at the MSIL level is as effective as computation of metrics directly on source code.

6. Conclusions and Future Work

The current trend towards the use of multiple languages in the development of software systems introduces new challenges for software comprehension and software maintenance. Many techniques and tools for clone detection, elimination, and removal have been described in the literature, and some of this literature addresses the problem of applying these techniques and tools to software systems written in a variety of languages. However, none of these techniques or tools have been applied to multi-language software systems with a focus on detecting code clones that span multiple languages.

In this paper we introduced an approach for detecting code clones that span multiple languages, which we termed *cross-language code clones*. Our approach complements other clone detection techniques, as it is focused on issues of detecting cross-language code clones and not on the mechanics of clone detection. We described an approach for cross-language clone detection and presented a tool, C2D2, that implements cross-language clone detection for the .NET Framework. Our experimental results demonstrate the existence of cross-language code clones, specifically code clones that span C# and Visual Basic.NET.

As future work we propose to enhance the usability and interoperability of C2D2 by producing output files that can be read by existing clone visualization systems, to integrate more advanced tree-based clone detection techniques into C2D2, and to perform more extensive studies of multi-language software systems.

References

[1] B. S. Baker. On finding duplication and near-duplication in large software systems. In *WCRE*, pages 86–95, July 1995.

[2] B. S. Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SICOMP*, 26(5):1343–1362, Oct. 1997.

[3] M. Balazinska, E. Merlo, M. Dagenais, B. Lague, and K. Kontogiannis. Partial redesign of java software systems based on clone analysis. In *WCRE*, pages 326–336, October 1999.

[4] H. A. Basit, D. C. Rajapakse, and S. Jarzabek. Beyond templates: A study of clones in the STL and some general implications. In *ICSE*, pages 451–459, May 2005.

[5] I. D. Baxter, A. Yahin, L. M. de Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *ICSM*, pages 368–377, November 1998.

[6] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo. Comparison and evaluation of clone detection tools. *TSE*, 33(9):577–591, Sept. 2007.

[7] Code Document Object Model. http://msdn2.microsoft.com/library/system.codedom.aspx.

[8] S. Ducasse, M. Rieger, and S. Demeyer. A language independent approach for detecting duplicated code. In *ICSM*, pages 109–118, August/September 1999.

[9] W. Evans, C. Fraser, and F. Ma. Clone detection via structural abstraction. In *WCRE*, pages 150–159, October 2007.

[10] R. Ferenc, S. E. Sim, R. C. Holt, R. Koschke, and T. Gyimothy. Towards a standard schema for C/C++. In *Proceedings of the 8th Working Conference on Reverse Engineering, Stuttgart, Germany*, pages 49–58. IEEE Computer Society, Oct. 2001.

[11] R. Geiger, B. Fluri, H. C. Gall, and M. Pinzger. *Relation of Code Clones and Change Couplings*, volume 3922 of *LNCS*, pages 411–425. 2006.

[12] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue. ARIES: Refactoring support environment based on code clone analysis. In *SEA*, pages 222–229, November 2004.

[13] L. Jiang, G. Misherghi, Z. Su, and S. Glondu. DECKARD: Scalable and accurate tree-based detection of code clones. In *ICSE*, pages 96–105, May 2007.

[14] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *TSE*, 28(7):654–670, July 2002.

[15] C. Kapser and M. Godfrey. "Cloning considered harmful" considered harmful. In *WCRE*, pages 19–28, October 2006.

[16] M. Kim, V. Sazawal, D. Notkin, and G. Murphy. An empirical study of code clone genealogies. *SEN*, 30(5):187–196, Sept. 2005.

[17] R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In *SAS*, pages 40–56, July 2001.

[18] K. Kontogiannis, P. Linos, and K. Wong. Comprehension and maintenance of large-scale multi-language software applications. In *ICSM*, pages 497–500, September 2006.

[19] K. A. Kontogiannis, R. Demori, E. Merlo, M. Galler, and M. Bernstein. Pattern matching for clone and concept detection. *JASE*, 3(1/2):77–108, July 1996.

[20] R. Koschke, R. Falke, and P. Frenzel. Clone detection using abstract syntax suffix trees. In *WCRE*, pages 253–262, October 2006.

[21] N. A. Kraft, B. A. Malloy, and J. F. Power. An infrastructure to support interoperability in reverse engineering. *Information and Software Technology*, 49(3):292–307, Mar. 2007.

[22] J. Krinke. Identifying similar code with program dependence graphs. In *WCRE*, page 301, October 2001.

[23] T. C. Lethbridge, S. Tichelaar, and E. Ploedereder. *The Dagstuhl Middle Metamodel: A Schema for Reverse Engineering*, volume 94 of *Electronic Notes in Theoretical Computer Science*, pages 7–18. Elsevier B.V., May 2004.

[24] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[25] Z. Li, S. Myagmar, and Y. Zhou. CP-Miner: Finding copy-paste and related bugs in large-scale software code. *TSE*, 32(3):176–192, Mar. 2006.

[26] P. Linos, W. Lucas, S. Myers, and E. Maier. A metrics tool for multi-language software. In *SEA*, pages 209–218, November 2007.

[27] P. K. Linos, Z. Chen, S. Berrier, and B. O'Rourke. A tool for understanding multi-language program dependencies. In *IWPC*, page 64, May 2003.

[28] J. Mayrand, C. Leblanc, and E. Merlo. Experiment on the automatic detection of function clones in a software system using metrics. In *ICSM*, pages 244–253, November 1996.

[29] Microsoft .NET Framework. http://msdn.microsoft.com/netframework/.

[30] D. L. Moise and K. Wong. Extracting and representing cross-language dependencies in diverse software systems. In *WCRE*, pages 209–218, November 2005.

[31] Mono. http://www.mono-project.com/.

[32] M. Rieger. *Effective Clone Detection Without Language Barriers*. PhD thesis, University of Bern, Switzerland, 2005.

[33] C. K. Roy and J. Cordy. Scenario-based comparison of clone detection techniques. In *ICPC*, June 2008. To appear.

[34] SharpDevelop IDE. http://www.sharpdevelop.net/.

[35] D. Strein, H. Kratz, and W. Löwe. Cross-language program analysis and refactoring. In *SCAM*, pages 207–216, September 2006.

[36] D. Strein, R. Lincke, J. Lundberg, and W. Löwe. An extensible meta-model for program analysis. *IEEE Transactions on Software Engineering*, 33(9):592–607, Sept. 2007.

[37] R. Tairas and J. Gray. Phoenix-based clone detection using suffix trees. In *ACMSE*, pages 679–684, March 2006.

[38] M. Toomim, A. Begel, and S. L. Graham. Managing duplicated code with linked editing. In *VL/HCC*, pages 173–180, September 2004.

[39] V. Wahler, D. Seipel, J. Wolff, and G. Fischer. Clone detection in source code by frequent itemset techniques. In *SCAM*, pages 128–135, September 2004.

Software Maintenance Maturity Model(S3^{m}DSS) A Decision Support System

Alain April[1], Naji Habra[2], Arnaud Counet[2]
(1)École de Technologie Supérieure, Montréal, Canada
(2)Faculté Universitaire Notre-Dame de la Paix, Namur, Belgium

aapril@ele.etsmtl.ca ,nha@info.fundp.ac.be , acounet@info.fundp.ac.be

Abstract - Maintaining and supporting the software of an organization is not an easy task, and software maintainers do not currently have access to decision support systems (DSS) to evaluate strategies for improving the specific activities of software maintenance. This article presents a DSS which helps in locating best practices offered by a software maintenance maturity model (S3^{m}). The contributions of this paper are: 1) to instrument a maturity model with a DSS tool to aid software maintenance practitioners in locating specific best practices that could help them answer their questions.

I. INTRODUCTION

Knowledge transfer of a large number of best practices, described in a maturity model, has proved difficult [1]. This is especially true during the training stage of an assessor or a new participant in a process improvement activity. It is also challenging to quickly refer to, or access, the right practice, or subset of practices, when trying to answer specific questions during or after a process maturity evaluation.

The software maintenance maturity model **S3^{m}** contains a large number of software maintenance concepts and information which are structured in many successive levels [2], [14]. The first level is labelled 'process domains level', and regroups the maintenance practices in 4 process domains (*process management*, *maintenance request management*, *software evolution engineering* and *support to software engineering evolution*). Each process domain is broken down into one or more key process areas (KPAs). These KPAs logically group together items which conceptually belong together. As an example all training related practices are grouped into one KPA. A KPA is further divided into *roadmaps* with one or more *best practices*, spanning five maturity levels. The complete **S3^{m}** has 4 domains, 18 KPAs, 74 roadmaps and 443 best practices. It would be beneficial to have a decision support system (DSS) to help access this complex structure and large amount of information. A potential solution to this problem would be to develop a decision based system for the **S3^{m}**. This DSS could be available for both maintainers and maintenance clients. The proposed modelling of a software maintenance DSS was based on the van Heijst methodology [3], which consists of constructing a task model, selecting or building an ontology [4], mapping the ontology onto the knowledge roles in the task model and instantiating the application ontology with this specific domain knowledge. According to van Heijst, there are at least six different types of knowledge to be taken into account when constructing such a system: tasks-goals, problem-solving methods, task instances, inferences, the ontology and the domain knowledge (see Fig.1). Van Heijst uses the different types of knowledge in a more generic way than we do in this document.

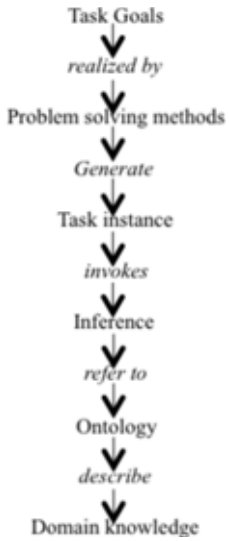


Fig.1. The different components of knowledge models [3]

For van Heijst, domain knowledge refers to a collection of statements about the domain [4]. The domain of this specific research is software maintenance, and it is divided into 4 process domains. Examples of statements are presented in section 3. At a high level, the ontology refers to a part of the software maintenance ontology [5] presented in section 4. The problem solving methods and tasks are described at length in section 5. The tool environment and conclusion, as well as future work, are presented in sections 6 and 7. Section 2 begins by presenting the goals of the **S3^{m}** architecture.

II GOALS OF THE *S^{3M}* ARCHITECTURE

The **S3^{m}** was designed as a customer-focused benchmark for either:

- Auditing the software maintenance capability of a service supplier or outsourcer; or
- Supporting the process improvement activities of software maintenance organizations.

To address the concerns specific to the maintainer, a distinct maintenance body of knowledge is required. The **S3^{m}** is also designed to complement the maturity model developed by the SEI at Carnegie Mellon University in Pittsburgh [6] by focusing mainly on practices specific to software maintenance. The architecture of the model locates the most fundamental practices at a lower level of maturity, whereas the most advanced practices are located at a higher level of maturity. An organization will typically mature from the lower to the higher maturity level as it improves. Lower-level practices must be implemented and sustained for higher-level practices to be achieved.

III *S^{3M}* ARCHITECTURE AND KNOWLEDGE STATEMENTS

Software maintainers experience a number of problems. These have been documented and an attempt made to rank them in order of importance. One of the first reported investigations was conducted by Lientz and Swanson [7]. They identified six problems related to users of the applications, to managerial constraints and to the quality of software documentation. Other surveys have found that a large percentage of the software maintenance problems reported are related to the software product itself. This survey identified complex and old source code which was badly documented and structured in a complex way. More recent surveys conducted among attendees at successive software maintenance conferences [8] ranked perceived problems in the following order of importance (see Table 1). These are also examples of knowledge statements about the domain of software maintenance. Key to helping software maintainers would be to provide them with ways of resolving their problems by leading them to documented best practices.

TABLE I
TOP MAINTENANCE PROBLEMS [8]

Rank	Maintenance problem
1	Managing fast-changing priorities
2	Inadequate testing techniques
3	Difficulty in measuring performance
4	Missing or incomplete software documentation
5	Adapting to rapid changes in user organizations
6	A large number of user requests in waiting
7	Difficulty in measuring/demonstrating the maintenance team's contribution
8	Low morale due to lack of recognition
9	Not many professionals in the field, especially experienced ones
10	Little methodology, few standards, procedures or tools specific to maintenance
11	Source code complex and unstructured
12	Integration, overlap and incompatibility of systems
13	Little training available to personnel
14	No strategic plans for maintenance
15	Difficulty in meeting user expectations
16	Lack of understanding and support from IT managers
17	Maintenance software running on obsolete systems and technologies
18	Little will for reengineering applications
19	Loss of expertise when employees leave

There is a growing number of sources where software maintainers can look for best practices, a major challenge being to encourage these sources to use the same terminology, process models and international standards. The practices used by maintainers need to show them how to meet their daily service goals. While these practices are most often described within their corresponding operational and support processes, and consist of numerous procedures, a very large number of problem-solving practices could be presented in a DSS which would answer their many questions about those problems. Examples are presented in section 6. Maintenance client problems could also be linked to these internal problems because of the impacts it can occur. When using the software maintenance ontology in the DSS, it was necessary to consider the structure of the maturity model relationship between the many process domains, roadmaps and practices. This problem is addressed next.

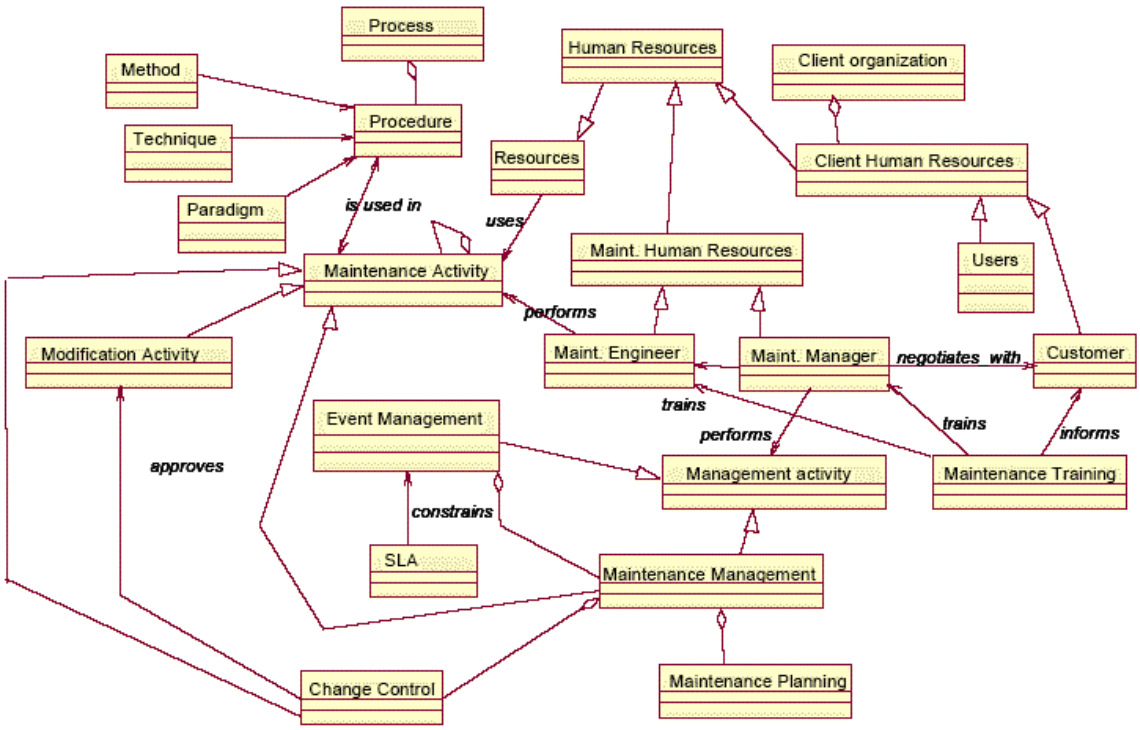


Fig.2. Part of the software maintenance ontology of (Kitchenham and et al., 1999)

IV ONTOLOGY OF THE SOFTWARE MAINTENANCE BODY OF KNOWLEDGE

We elected to implement only a subset of the ontology developed by Kitchenham et al. [5] and Ruiz et al. [9] for the initial trial of this research project. The Kitchenham ontology was chosen because its author is well known in Software Engineering maintenance. Other software maintenance ontologies could also be used [9], [10] and [11] to enhance the Kitchenham at al. proposal. Fig. 2 describes the different maintenance concepts considered surrounding a software maintenance activity. Software maintenance is highly event-driven, which means that some maintenance activities are unscheduled and can interrupt ongoing work. This subset of the ontology represents many, but not all, the concepts involved in responding to the questions related to the first problem identified by Dekleva [8]: "Managing fast-changing priorities". Maintainers agree that this is the most important problem they face. How can they handle the fast-changing priorities of the customer? Solutions to this problem are likely to be found by using many paths through the maintenance concepts of the ontology. Navigation through these concepts should lead to associated concepts which are conceptually linked and likely to contribute to a solution, like the need for better event management, change control, maintenance planning, Service Level Agreements, maintenance manager negotiation, training, procedures, and so forth. Many more concepts must be involved to contribute to all aspects of the solution, but our purpose is to show the utility of a DSS in the software maintenance domain, and it therefore starts with a constrained number of concepts. Maturity models typically include the detailed best practices that could be of help in solving this type of problem. The main issue is that the best practice locations and their interrelationships are hidden in the layered architecture of the maturity model, specifically in its process domains, KPAs and roadmaps. It is therefore necessary to find a way to link this layered architecture with the maintenance concepts of the ontology and proceed to analyze the tasks required to build a DSS to support the maintainers in their quest for solutions. The next section describes the navigation concepts that have been implemented in ***S3^{m}DSS***. The user of the DSS navigates using a sequence of tasks that will lead him through a further sequence of tasks.

V HIGH LEVEL VIEW OF S^{3M} DSS

In [3], the first activity in the construction of a DSS is the definition of task analysis. Task analysis begins, at a high level, with a definition of an index of terms. This index includes words commonly used in software engineering (see Figure 3). From this index, a subset of

more restrictive words is identified. This subset is a list of keywords recognized specifically in software maintenance. Each keyword is then connected to one or more maintenance concepts. A maintenance concept, in software maintenance, is a concept found in the Software Maintenance Body of Knowledge and ontology (see Fig. 2). Every maintenance problem identified by Dekleva [8] has been translated into a case problem and connected to the software maintenance ontology. Each case problem is then linked to a set of themes (questions) which help the user of the DSS to navigate into a part of the maturity model that will propose recommendations in the form of best practices. The link between the maintenance concepts and the maturity model is made in the themes concept. Themes are questions which have been developed to hop from node to node in the ontology. A close look at Fig. 2 reveals that the themes concept can combine different maintenance concepts and, finally, create a set of recommendations of the maturity model. For every best practice, there is a linked theme (or choice) from which the user can select (also called facts) which will lead to a final specific set of recommendations. This 1-1 matching between theme and recommendation will contribute to a composed set of recommendations directly adapted to the user context. Above all of this, a distinction between internal maintenance engineers and maintenance client has been made. We think that the same problems are involved for both side but we need to adapt the way we ask. In this case, when a maintenance client uses the system, themes are adapted to his understanding.

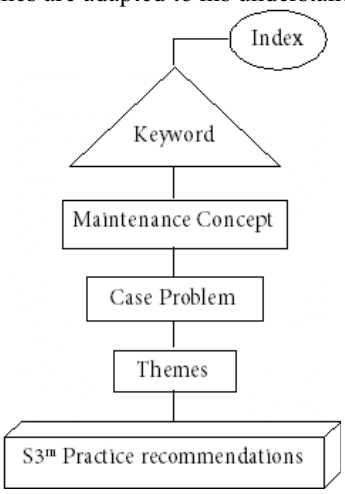


Fig.3. High-level view of S3^m

Provided recommendations are some kind of invitation to his maintainer to follow different rules. This could both help maintenance enterprise but also client one.

Expanding the 6 high-level tasks in Figure 3, we propose 12 detailed tasks which will help identify a subset of best practices related to the S3^m.

VI DSS TOOL TECHNOLOGY

Next we will explain the technology used as well as an overview of the design of the DSS. Then we will demonstrate how this DSS can be used to help a user answer a question and how an expert populates a complete case problem.

A. Technology and design

The ***S3^{m}DSS*** was built using Java, Java Server Pages, JavaScript, CSS and HTML technologies. This combination of technologies was selected for its easy access via the Internet.

TABLE II
DSS QUESTIONS

#	Questions
A	Are there training plans for to new maintenance engineers about generic topics like management and processes activities?
B	Do maintenance engineers periodically update their knowledge associated with the software and its infrastructure they maintain?
C	Are maintenance engineers trained and motivated to perform well when using the processes/services and their support role?
D	Is there some training communication with customers offered to software maintenance engineers?
E	Do you use any internal benchmarking data to guide the training of maintenance resources?
F	Does the maintenance organisation have a training budget?
G	Are there plans describing the training needed for each maintenance position and application software?
H	Is there training time planned?
I	Do senior maintainers familiarise new employees?
J	Are training needs defined for both technical and management responsibilities for each development project?
K	Do people working on the pre-delivery and transition receive the training deemed appropriate by the software developer?

Behind that, a SQL Server database was added in order to manage the knowledge base. This choice was justified by the lack of reactivity that XML parsing proposed before. The architecture is based on a 3-tiers

model providing easy maintainability and is composed of a presentation layer, a business layer and DAO layer. The business layer design has been split into 2 parts: the first part regroups all the controlling servlets and the second parts regroup all the business methods. Servlets assure proper communication between the presentation layer and the business layer while the business methods communicate with the DAO layer.Currently, more than 550 words and 70 keywords have been introduced into the DSS. Five maintenance problems identified by Dekleva have been introduced and took 17 hours to complete.

We estimate that there is still 2,000 hours required to populate the knowledge base for all the ***S3m*** practices for maturity 0, 1 and 2. The DSS has 3 different interface types: administrator, expert and user. The administrator interface manages access rights to the DSS, while the expert interface offers experts the option of adding new index words, keywords, concepts, cases, themes and recommendations. The next section will demonstrate how the DSS helps a user answer a specific question: How can I improve a maintainers training?

B. Helping a user answer a question

First of all, the user enters a word that will identify a suggested keyword that represents the topic he is interested to obtain answers for. As an example, the user enters: *training*. This keyword will guide the DSS to the most closely related KPA and roadmap concepts of its database. Currently, the DSS presents the following keyword: *maintenance training* as a feedback. In this same feedback the DSS presents the maintenance concepts, which are related to this KPA and roadmap, to the user.

It also presents the concepts in order of priority. This is done using a percentage of relevance linked to each concept. The expert had previously established this percentage. The user is then asked to choose one or multiple concepts, *maintenance human resources* in our example. The DSS presents the case problems associated with this maintenance concept selected to the user. It will present the case problems in order of priority to the user. A percentage of relevance is also related to each case problem. The expert has also previously established this percentage. The user chooses one or multiple case problems that represent the closest is current problem, ex: *little training available to maintenance engineers* in our example. With this case problem, there are 11 themes presented to the user in the form of questions (see Table 2). The user will find facts for each practice (theme). He can answer yes or no to any of the themes. In function of the facts chosen, the system composed a set of recommendations to the user. Figure 4 shows how the DSS will recommend the following solution (simplified for this paper): RecSet.

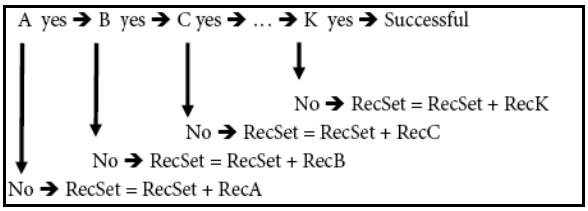


Fig. 4: DSS recommendation mechanism

Figure 5 (next page) shows an example of the user layout in the previous case problem. The user layout is made up of 4 dynamic tables representing all the concepts we discuss before.

Each table is displayed step by step by user selection and associated with a help function. In the top of the layout, a toolbar has been inserted to start every research by typing a word into the system or selecting a keyword. Next will show in practice how a maintenance expert can enter a case problem into the DSS.

VII EXPERT INTERFACE

Fig. 5 (next page) show an example of the expert layout. This layout asks experts to add, modify or delete high level view elements. Expert can also add a complete case to the DSS by respecting the following recommendation, question, case problem, maintenance concept, keyword and word order because of the links between elements. Below the top table, a form is proposed where expert can fill information like element name, help content or links with upper or lower elements.

All existing elements are accessible by conventional html lists and can be added very easily by selecting and pressing a button. When validation button is pressed, an additional form shown in Fig. 6 appear. Experts can then complete association percentages between linked elements. Note that experts can use HTML mark-ups into recommendation text to add hyperlinks, lists or tables.

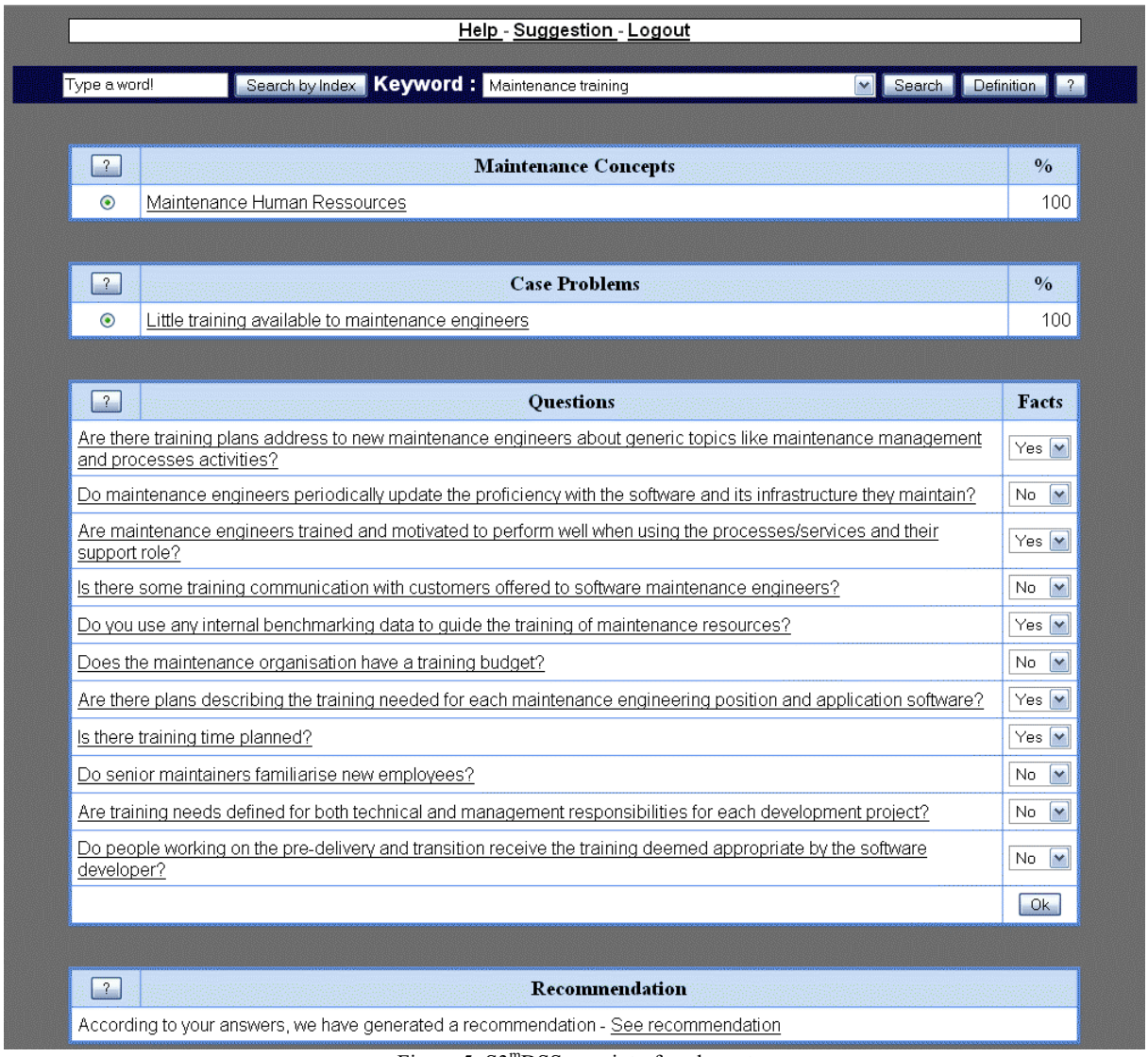


Figure 5: S3^{m}DSS user interface layout

X Conclusion and Future Work

Identifying the best practices in a maturity model is a difficult task, considering their number and the multiple possible answers associated with each of them. Our proposal is that a DSS could help in finding an appropriate recommendation. The next step in this research project is to populate the DSS, validate the results with experts in the domain and determine whether or not the DSS is a useful support tool for training on the content of the maturity model. The **S3^{m}DSS** is a working prototype and is available on **http://www.s3m.ca**. Future work will consist of first creating a higher level representation of the key users, customers, maintenance managers and maintenance engineers concerns. This will be helpful for users to navigate first in all the software maintenance problems before they can drill down to a specific area. Second tasks will be to enhance the number of maintenance problems and insert examples of how the case problem. A case problem is an example of what other companies have done to solve a specific issue. We have been tracking the usage of the DSS for 2 years now and can report on its usage. Although users will be able to find a recommendation there is little evidence that this information is helpful in their daily work. More validation is required to see if a DSS in this very unstructured and low maturity domain could yield any benefit to an organization. More research will be conducted this year with the help of master students from the FUNDP from Belgium.

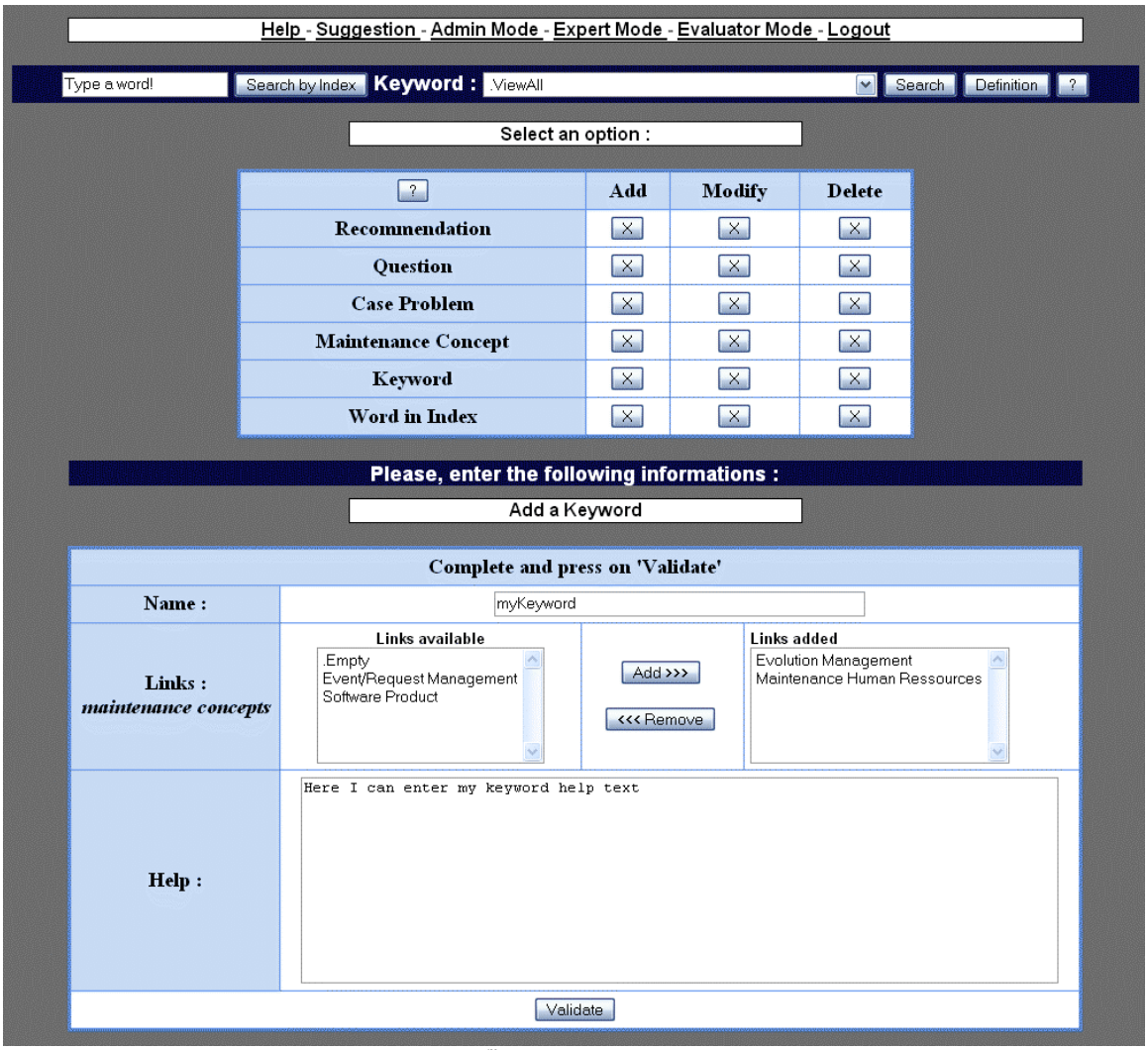


Figure 6: S3[m]DSS expert form layout

REFERENCES

[1] Abran, A., Moore, J. W., Bourque, P., Dupuis, R. and Tripp, L.,*Guide for the Software Engineering Body of Knowledge (SWEBOK),* Ironman version, IEEE Computer Society Press: Los Alamitos CA,2004; 6-1-6-15, Montréal, http://www.swebok.org [27 January 2005].

[2] A. April, J. Huffman Hayes, A. Abran, R. Dumke, "Software Maintenance Maturity Model (SMmm): The software maintenance process model", Journal of Software Maintenance and Evolution: Research and Practice, 17(3): May/June 2005:197−223.

[7] Lientz, B. and Swanson, E. (1981), *Problems in Application Software Maintenance*, Communications of the ACM, **24, 11,** 763-769.

[8]Dekleva, S. M. *Delphi Study of Software Maintenance Problems,* International Conference on Software Maintenance (CSM 1992) (1992) IEEE Computer Society Press: Los Alamitos CA

[9] Ruiz, F., Vizcaino, A., Piattini, M. and Garcia, F. (2004) *International Journal of Software Engineering and Knowledge Engineering,* **14, 3** 323-349.

[10] Vizcaíno, A. Favela, J. and Piattini, M. *A multi-agent system for knowledge management in software maintenance*, KES 2003 (2003), Springer Verlag, Oxford, UK.

[11] Dias, M., G. Anquetil, N. and Oliveira, K. M. (2003), *Organizing the Knowledge Used in Software Maintenance*, Journal of Universal Computer Science, **9, 7** 64-658.

[12] Counet, A. (2007), Mémoire de maîtrise, FUNDP, Namur, Belgium.

[13] Desharnais, J.-M., *Application de la mesure fonctionnelle COSMIC-FFP: une approche cognitive*, UQAM, Montréal, 2004

[14] April, A., Abran A. and Dumke, R. *Assessment of Software Maintenance Capability: A model and its Design Process*, IASTED 2004, Conference on Software Engineering (2004b), Innsbruck (Austria).

Odyssey-MEC: Model Evolution Control in the Context of Model-Driven Architecture

Chessman Corrêa Leonardo Murta Cláudia Werner

Federal University of Rio de Janeiro
COPPE - System Eng. and Computer Science
{chessman, murta, werner}@cos.ufrj.br

ABSTRACT
Model-Driven Development aims to use models as first class artifacts in software development. Therefore, the need to control model evolution in this context became as important as to control the evolution of source-code. In Model-Driven Architecture, a target model is generated from a source model through a transformation process. Consequently, there is a relationship among them. However, these models may evolve independently due to modifications, making them inconsistent with each other. In this scenario, traditional versioning is fundamental, but it is not sufficient to control the evolution of different interconnected models that represent the same software. In this paper, we propose a server side transformation, synchronization and versioning approach to control the evolution of models.

Keywords
Version Control, Model Versioning, Model-Driven Development, Model-Driven Architecture, Model Evolution.

1. INTRODUCTION
Model-Driven Architecture (MDA) is the Object Management Group (OMG) framework for **Model-Driven Development** (MDD) [18]. One characteristic of this approach is the generation of a target model from a source model using a transformation engine. It means that the software is represented by different models, most of the time in different abstraction levels.

In large software projects, multiple people assuming specific roles and located at different places may modify related models independently. For example, a PSM (Platform Specific Model) generated from a PIM (Platform Independent Model) may need to be modified because it does not have all the necessary details to derive the source-code. In other words, models may need to be updated in order to be used to generate other models or source code.

Since these models are related with each other, modifications applied to a model may create inconsistencies between them. However, as these models represent the same software, inconsistencies cannot be allowed. For example, inconsistencies between PIM and PSM introduce some difficulties to generate PSMs tailored to other platforms. This is especially true if PIM level changes are made in PSM instead. In this case, the generation of PSM to a new platform would not have PIM details that exist in the other platform. It means that if a MDD project aims to create software for different platforms, PSMs of each platform have to be consistent with the corresponding PIM and with PSMs of other platforms. It is also true for models in the same abstraction level.

Model versioning is essential to control model evolution. However, if source and target models are versioned independently, there will be no guarantee that they are consistent with each other. Since these models have to evolve together, versioning is not enough to control their evolution in MDA context. Therefore, these models have to be synchronized before versioned.

Models synchronization is achieved from round-trip engineering through bidirectional transformations that preserve previous versions of existing models. However, if a synchronization tool is not automatically executed, software engineers may forget to use them, leading to inconsistent models.

Based on these facts, this paper proposes a server side model transformation, synchronization and versioning approach to control model evolution in MDA.

The rest of this paper is organized as follows. Section 2 briefly describes the Model-Driven Architecture. Section 3 discusses the key aspects of our approach. Section 4 presents some related works. Finally, the conclusion and future work are presented in Section 5.

2. MODEL DRIVEN ARCHITECTURE
OMG was inspired by constantly shifting infrastructures, requirements changing, and new emerging technologies to create the **Model-Driven Architecture** (**MDA**) [18]. This approach considers models as first-class development artifacts and uses them not only for understanding and commu-

nication, but also for design, construction, deployment, operation, maintenance, and modification of a system.

2.1 MDA Models

MDA specifies four kinds of models: **Computation Independent Model** (CIM), **Platform Independent Model** (PIM), **Platform Model** (PM) and **Platform Specific Model** (PSM) [18]. **CIM** represents the system requirements. It takes into consideration domain concerns, such as the vocabulary used by the domain practitioners. CIM represents a view of the system without computational details. **PIM** is a view of the system considering computational solutions that aim to be generic to any platform. Thus, it represents a system that can be tailored to multiple platforms, assuming that these platforms are compatible to the architectural styles adopted in the corresponding PIM. **PM** provides the technical concepts, requirements, and services of a specific platform. **PSM** is a view of a system considering the platform details. It can be seen as a merge of PIM and PM, augmented by some changes specific to the target platform.

2.2 Model Transformation

Model transformation is the process of creating a target model from a source model of the same system. Although it could be made manually, the MDA approach aims at automating this operation. This is a key factor to the increasing MDA adoption over traditional software development.

In **forward engineering**, a model-to-model transformation uses the CIM and other information to generate a PIM. Subsequently, another model-to-model transformation combines PIM and PM to create the corresponding PSM. Finally, PSM is used by a model-to-text transformation to generate the source-code to the specific software platform.

A model transformation uses **mappings** to create target model elements from source model elements. Mappings provide specification of how one or more target elements are derived from source elements. It also may have mapping rules based on specific **marks**, like stereotypes and tagged values. For example, a PIM class with the stereotype <<*entity*>> may generate an EJB (Enterprise JavaBean) class for the JEE[1] platform.

During model generation, the model transformation should also generate the **record of transformation**. It includes the **traceability links** between source and target model elements and informs which parts of the mapping were used during the generation. It is an important resource to support synchronization.

It is important to notice that CIM, PIM and PSM are in different abstraction levels. This mean that CIM-PIM and PIM-PSM transformations are vertical transformations among different abstraction levels. However, it is also possible to generate models in the same abstraction level through horizontal transformations, such as PIM-PIM and PSM-PSM.

2.3 MDA Application

The application of MDA is relatively simple. It can be divided into two main phases: infrastructure setup and transformation. The infrastructure setup starts with the creation of mappings based on a platform or a set of platforms. These mappings will be used by transformations. In addition, the marks to be applied on a PIM may also be defined, usually through a Profile [19].

After this initial setup, the software engineer uses a modeling tool to create a model (e.g., a PIM). Afterwards, that model may be marked according to the available Profiles. Finally, the transformation is executed, using the mappings and the marked model to generate the corresponding model (e.g., a PSM) and the record of transformation.

This scenario focuses on forward engineering. However, it is also possible to occur reverse engineering transformations, generating a PIM from a PSM.

3. ODYSSEY-MEC

In this section we introduce Odyssey-MEC (Odyssey for Model Evolution Control), a server side transformation, synchronization and versioning approach to control MDA models evolution.

In the following, we detail our approach presenting its architecture, model infrastructure, model repository, model versioning, model transformation, record of transformation, element search, and model synchronization.

3.1 Architecture

The architecture of the approach is shown in Figure 1. It has four types of repositories: Transformation Mappings, PIM, PSM, and Record of Transformation. The **Transformation Mappings Repository** (TMR) stores the transformation mappings to be used by the transformation engine. These transformation mappings are created by a transformation engineer, as specified by Bacelo et al [1]. **PIM and PSM Repositories** store PIMs and PSMs, respectively. It is worth to notice that these repositories persist versioned models. Moreover, each platform has its own PSM repository. **Record of Transformation Repository** (RTR) stores the Record of Transformations (RT).

Our approach comprises three main components to control model evolution: Odyssey-VCS [15, 17], Odyssey-MDA [1], and a synchronization engine (SE). It also uses a Transaction Manager (TM) component to control the synchronization and versioning process in a transaction context. The

[1] http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

Odyssey-VCS component is used for model versioning and the Odyssey-MDA component for model transformation.

Odyssey-VCS has hooks that execute the TM and the SE when a model is checked in. SE uses Odyssey-MDA to generate target models, and Odyssey-VCS to access the models to be synchronized and their versioning data. It also uses RT as an auxiliary resource to synchronize the models. Finally, an Odyssey-VCS client is used to communicate with Odyssey-VCS server (it can be any CASE tool that exports models through XMI 2.1 format). Odyssey-VCS client communicates with Odyssey-VCS server through Web Services [4].

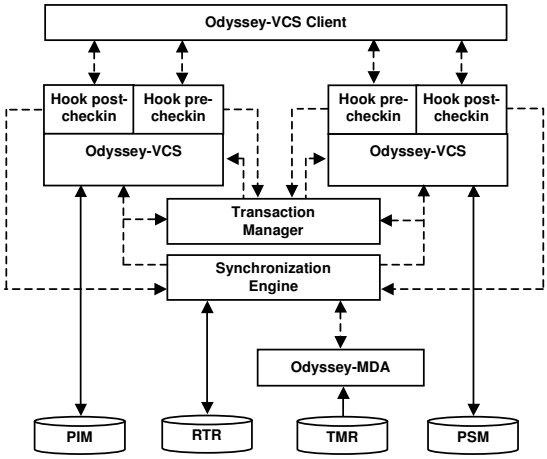


Figure 1. Odyssey-MEC Architecture

3.2 Model Infrastructure

OMG chose UML (Unified Modeling Language) as the standard modeling language for MDA. Therefore, our approach controls the evolution of UML models.

Although OMG uses MOF (Meta Object Facility) as UML meta-model, Odyssey-MEC uses the Eclipse Ecore meta-model [5]. The use of Ecore instead of MOF is not an obstacle to control the evolution of UML models because EMF uses XMI (XML Metadata Interchange) [20] for externalizing UML models. Client tools just have to use the same XMI version used by EMF (version 2.1).

3.3 Model Repository

Model repositories are used to store models. In our case, it is necessary to store all versions of a model to control the model evolution.

Due to the lack of versioning repositories for EMF, we adopted Odyssey-VCS as our versioning component, as detailed in Section 3.4.

3.4 Version Control

Version control is a key resource to control the evolution of models during development and maintenance. It is used to generate a history of model versions and maintain information like when, why, and who has made modifications. This history of model versions and modification information are stored in a repository. The basic functionalities of version control systems are: check-in (save a model into the versioned repository), check-out (get a model from the versioned repository), merge (join two models) and detect conflicts (identify concurrent modifications that cannot be resolved)[2].

Our model-based version control component is Odyssey-VCS. This component has a client/server architecture and offers all the requirements discussed above. It uses the EMF reflective API to support the versioning of any UML model element[2]. It can also execute external code trough hook implementation.

Odyssey-VCS works at fine-grained model versioning. This means that it is capable of identifying a new version of any UML model element. When a model element is composed from other model elements, if one of these elements is changed, the composing model element also receives a new version number. This is propagated recursively up to the outer model element, frequently a model package.

3.5 Model Transformation

Model transformation depends on a set of mappings and rules to create elements in a model from elements of another model. There are different ways to generate a new model using transformations [18]. Some existing approaches to model-to-model transformations are: ATL (ATLAS Transformation Language) [11], Triple [3], OptimalJ [7]. UMT [16], UMLX [8], and Odyssey-MDA[1]. A further discussion about transformation can be found in [21].

One of the requirements for controlling model evolution is the support for bidirectional transformations. This means that transformations should be able to generate PSM elements from PIM elements, and PIM elements from PSM elements. This feature is needed because different people may be working over different models, and new elements inserted in a PSM may have to be represented in its corresponding PIM. From the approaches presented above, ATL [11] and Odyssey-MDA [1] allow transformations in both directions. However, ATL requires the writing of a particular transformation mapping for each direction. On the other hand, Odyssey-MDA allows the specification of bidirectional transformations in the same mapping. In addition, it is also shipped with a tool for model marking, named ModelMarker. Due to that, we adopted Odyssey-MDA as our transformation engine component.

[2] A model element is any UML element defined in its metamodel, for example, a class, attribute, operation, component, association, etc.

Odyssey-MDA is capable to execute vertical and horizontal transformations. Therefore, although this paper is focused in vertical transformations, our approach can also be applied to control the evolution of models at the same abstraction level.

3.6 Record of Transformation

A record of transformation (RT) [18] is used to identify source models from target models and vice-versa. This is a very important resource to synchronize models, as it helps to identify existing model elements that have to be updated instead of being overwritten. Traceability links are particularly important for the synchronization activity when some relevant information is lost during the transformation [22].

In Odyssey-MEC approach, RTs are represented as a Traceability Links (TL). This is an Ecore model element that we created to reference the source and target model elements and the mapping that was used to generate the target element. Traceability links are generated by our Odyssey-MDA component during model transformations. Since more than one source or target model may be involved in transformation, it is possible that more than one traceability link references the same source element or target element.

3.7 Transaction Control

The synchronization and versioning of source and target models should be performed in a transaction context. In other words, if one of these steps fails, the whole process has to be canceled to avoid model inconsistencies.

To solve this problem, we adopted a Transaction Manager (TM) component that implement a two-phased transaction commit. When a model is checked in, the Odyssey-VCS pre-checkin hook uses TM to verify if there is any existing transaction in progress. If not, it asks for a new transaction and informs Odyssey-VCS that the model can be versioned. Odyssey-VCS starts its own transaction to create the model version. After versioning the model, Odyssey-VCS executes the post-checkin hook. This hook initiates the transformation and synchronization process. During this activity, other Odyssey-VCS instances may start their own transactions, as well as RTR. If all Odyssey-VCS instances finish their transactions successfully, TM navigates trough all Odyssey-VCS instances asking them to confirm their transactions. This also happens with RTR. Finally, the global transaction is confirmed.

3.8 Element Search

The versioning process depends on finding prior element versions. The synchronization process depends on finding PIM and PSM elements that have a trace relationship. Due to that, our element search occurs in two dimensions: time (different versions) and space (different models).

UML model elements are identified in XMI files by unique identifiers. Unfortunately, most tools do not preserve the value of these ids when models are exported. Therefore, this identifier cannot be used do identify model elements. To solve this problem, Odyssey-VCS uses a unique identifier as a tagged value.

The Odyssey-VCS meta-model has an element called Version. This element represents a version of a UML model element, and stores some versioning data, such as the element version number. It also has references to the UML model element it represents and references to the prior and next versions. Therefore, there is a list of versions for each element, which constitutes the element version history. This version history is useful to find prior and next versions of an element. However, due to the use of separate repositories for PIM and PSM, elements in different models have their own version history.

The combination of version history list and traceability links can be seen in Figure 2. Together, these two references make it possible to freely navigate from one version to another and from an element of a model (e.g., PIM) to another element of another model (e.g., PSM). This capability supports the versioning and synchronization processes discussed in Section 3.9.

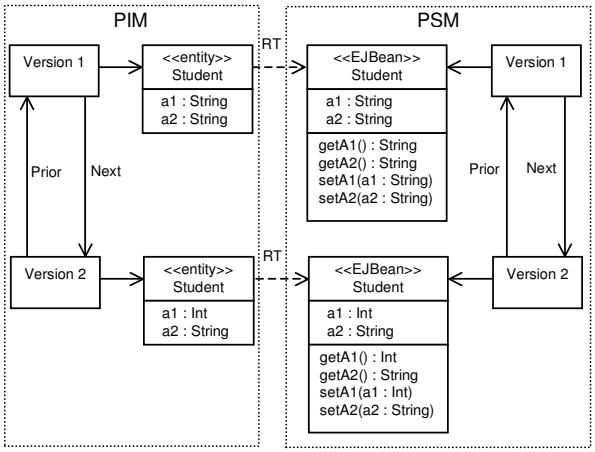


Figure 2. Version, PIM and PSM references

3.9 Model Synchronization

Interrelated models have to be consistent with each other. Therefore, it is necessary to synchronize them during development and maintenance, but preserving prior modifications. The ability to automatically synchronize models without information loss is called **roundtrip engineering** [22], and the lack of this ability usually leads to legacy systems [13].

Odyssey-VCS is designed to control the evolution of independent models. It means that this component alone is not capable of controlling the evolution of models that have

traceability links among them. Therefore, it is necessary to adopt a synchronization engine together with Odyssey-VCS. This synchronization engine is triggered by Odyssey-VCS hooks.

The synchronization engine, which is a component of Odyssey-MEC, depends not just on PIM and PSM version control information, but also on existing record of transformations of prior PIM and PSM versions. It also depends on Odyssey-MDA to generate models.

When a model is checked in, Odyssey-VCS tries to create a new version of the model. The Synchronization Engine (SE) is executed only if there is no version conflict during the versioning process. This avoids synchronization effort in cases of conflicts. If there is no conflict, SE selects the transformation mapping to be used and sends it to Odyssey-MDA, together with the new model version that was checked-in. Odyssey-MDA generates the target model (TM) and the RT of each target element, and returns them to SE. SE uses Odyssey-VCS of the target element to verify if there is any existing version available. If no previous version is found, SE considers the target model as a new model. In this case, SE checks in the target model using the Odyssey-VCS repository designated to it.

If there is an existing version of the generated target model, SE has to pre-process it in order to allow Odyssey-VCS to match the model with its prior version. This pre-processing starts with the recovery of versioning information. After that, the versioning information is interwoven into the generated target model.

This pre-processing process is composed of the following steps: (1) SE navigates trough all elements of the generated target model; (2) Using the traceability link, SE finds the related source model; (3) SE searches for the most recent version that has a traceability link dependency to an element of the target model; (4) When this element is found, SE retrieves its version information and puts into the respective generated target model element.

After the process is finished, SE checks in the model. It is worth to notice that, at this moment, the generated target model has all the necessary versioning information to allow Odyssey-VCS to interpret it as a new version of a model under version control. The generation of a new version of the model element means that the differences between the existing version and the checked-in version were merged. In other words, the synchronization was performed. If some conflicts occur during this merge process, all the operations are canceled.

When a source element has traceability links to more than one target element, the part of the transformation used to generate the target element is used to identify the correct element. This information is specified together with the traceability link that exists between the source and target models.

4. RELATED WORK

Gîrba et al. [10] proposes Hismo as a meta-model based solution to control model versions. However, this approach does not take into consideration synchronization and does not support UML models.

Matheson et al. [14] proposed an architecture for capturing models evolution in MDD. They suggest the use of a repository centric solution that is independent from client tools and stores model versions and their relationships in fine granularity. XMI is proposed as the data exchange mechanism for UML artifacts, and it uses XML and XML Schema to specify the transformation specifications. Besides the similarities with our approach, nothing was mentioned about the execution of model synchronization and model versioning.

There are some other researches [6, 9, 12] that take model evolution into consideration in some different ways, but do not consider versioning and model synchronization, as we do in our work.

5. CONCLUSIONS

This paper presented an approach to control the evolution of MDA models considering the model synchronization and versioning in a client/server architecture. Therefore, any CASE tool that can export models using XMI format is a potential client to Odyssey-MEC.

The way that PIM and PSM are versioned in Odyssey-MEC eliminates the need of any special mechanism to synchronize them. This synchronization is made when Odyssey-VCS merges the model that is being checked in with its last available version.

The client/server architecture of Odyssey-MEC makes it possible to implement distributed MDD using the MDA approach. The automatic model synchronization avoids the errors that can be introduced during manual synchronization. It also guarantees that models will always be consistent.

Although this paper focused on PIM and PSM, models in the same abstraction level may also be generated, synchronized and versioned. This can be done via horizontal transformation definitions during the MDD project creation. Moreover, we were mostly focused in this paper on PIM and PSM synchronization and versioning. Therefore, only two abstraction levels where considered. However, the approach works with unlimited abstraction levels. In this case, a PSM can be considered a PIM for the next abstraction level. It is also possible to support PSM for multiple platforms.

Currently, Odyssey-MDA works just with static models (i.e., class and component models). It means that Odyssey-

MEC cannot synchronize dynamic models, such as sequence model. Nevertheless, Odyssey-VCS can still be used to version control these models, but without synchronization among them. Moreover, the current version of Odyssey-MDA is able to deal with just one model as input and generates another model as output. Therefore, Odyssey-MEC supports model evolution in a one-to-one basis.

Our next step is to evaluate the proposed approach by applying some selected cases that will take into consideration conflict resolutions, forward and reverse transformations, transformation mapping change, etc. The results will be evaluated through precision and recall analysis [23], comparing them to the expected values.

As future work, we intend to: (1) expand our support to CIM and source-code; (2) develop an additional tool to help de visualization of MDA models evolution during the project execution and system maintenance; (3) control the evolution of transformation mappings and register in the RT the version of the transformation mapping used during the transformation; (4) expand our support to other UML models, such as the behavioral models; (5) modify Odyssey-MDA to receive and generate more than one model; and (6) use rules do control modifications that can be applied on interrelated models.

6. ACKNOWLEDGMENTS

Our thanks to the members of the Software Reuse Group at COPPE/UFRJ, especially Hamilton Oliveira, Cristine Dantas, Luiz Gustavo Lopes, João Gustavo Prudêncio, and Natanael Maia, who contributed to Odyssey-VCS and Odyssey-MDA. We also want to thank CNPq for the financial support.

7. REFERENCES

1. Bacelo, A., Maia, N. and Werner, C.M.L., Odyssey-MDA: A Transformational Approach to Component Models. in *Proceedings of Conference on Software Engineering and Knowledge Engineering*, (Boston, USA, 2007), 9-14.
2. Berczuk, S. *Software Configuration Management Patterns: Effective Teamwork, Practical Integration* Addison-Wesley, Boston, MA, USA, 2002.
3. Billig, A., Busse, S., Leicher, A. and Süb, J.G., Platform Independent Model Transformation Based on Triple. in *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, (Toronto, Canada, 2004), 493-511.
4. Booth, D., Hass, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. and Orchand, D. Web Services Architecture - W3C Working Group Note, World Wide Web Consortium (W3C), 2005.
5. Budinsky, F., Steiberg, D., Merks, E., Ellersick, R. and Grose, T.J. *Eclipse Modeling Framework: A Developer's Guide*. Addison Wesley, 2003.
6. Chen, F., Yang, H., Qiao, B. and Chu, W.C.-C., A Formal Model Driven Approach to Dependable Software Evolution. in *Proceedings of 30th Annual International Computer Software and Applications Conference - Cover*, (Chicago, Illinois, USA, 2006), 205 - 214.
7. Compuware. OptimalJ - Model-driven Java Development Tool, 2007.
8. Eclipse. UMLX A Graphical Transformation Language for MDA, 2007.
9. Engels, G., Küster, J.M., Heckel, R. and Groenewegen, L., Towards Consistency-Preserving Model Evolution in *Proceedings ICSE Workshop on Model Evolution*, (Florida, USA, 2002), 129-132.
10. Girba, T., Favre, J.-M. and Ducasse, S.e. Using Meta-Model Transformation to Model Software Evolution. *Electronic Notes in Theoretical Computer Science*, *137*. 57-64.
11. Jouault, F. and Kurtev, I., Transforming Models with ATL. in *Proceedings of the Model Transformation in Practice Workshop at MoDELS*, (Montego Bay, Jamaica, 2005), 128-138.
12. Lin, Y. and Gray, J., A Model Transformation Approach to Automatic Model Construction and Evolution. in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, (Long Beach, CA, USA, 2005), ACM, 448-451.
13. Maciaszek, L.A., Roundtrip Architectural Modeling. in *Proceedings of the 2nd Asia-Pacifc Conference on Conceptual Modeling*, (Newscastle, Australia, 2005), Australian Computer Society, Inc. , 17-23.
14. Matheson, D., France, R., Bieman, J., Alexander, R., DeWitt, J. and McEachen, N., Managed Evolution of a Model Driven Development Approach to Software-based Solutions. in *Workshop on Best Practices for Model Driven Development*, (Vancouver, Canada, 2004).
15. Murta, L.G.P., Dantas, H.L.R., Lopes, L.G.B. and Werner, C.M.L. Odyssey-SCM: An Integrated Software Configuration Management Infrastructure for UML Models. *Science of Computer Programming*, *65* (3). 249-274.
16. Oldevik, J. UML Model Transformation Tool - Overview and User Guide Documentation, 2004.
17. Oliveira, H., Murta, L. and Werner, C.M.L., Odyssey-VCS: a Flexible Version Control System for UML Model Elements. in *International Workshop on Software Configuration Management (SCM-12)* (Lisbon, Portugal, 2005), 1-16.
18. OMG. MDA Guide Version 1.0.1, Object Management Group, 2003.
19. OMG. Unified Modeling Language (UML) Infrastructure Specification. Version 2.0, Object Management Group, 2006.
20. OMG. XML Metadata Interchange (XMI) Specification. Version 2.0, Object Management Group, 2005.
21. Sendall, S. and Kozaczynski, W. Model Transformation - the Heart and Soul of Model-Driven Development. *IEEE Software*, *20* (5). 42-45.
22. Sendall, S. and Küster, J., Taming Model Round-Trip Engineering. in *Workshop on Best Practices for Model-Driven Software Development*, (Vancouver, Canada, 2004).
23. Yates, R.B. and Neto, B.R. *Modern Information Retrieval*. ACM press, 1999.

Analyzing the Impact of Attribute Noise on Software Quality Classification

Andres A. Folleco
Taghi M. Khoshgoftaar*
Lofton A. Bullard

Abstract

A ubiquitous problem in software quality classification is the presence of noise in measurement data. Noise can have a tremendous effect on classification performance. Relatively few studies, and none in the software quality estimation domain, have considered the impact of noisy attributes on classifier performance. This study investigates the impact of attribute noise on the performance of 11 software quality classification models. Noise was injected into seven real-world software measurement datasets, initially relatively free of noise. Attributes were ranked based on their KS statistic to determine their predictive significance. Those with higher rankings were injected with noise, in order of decreasing KS statistic. If the number of attributes injected with noise exceeded a threshold value found in this study, the classification performance deteriorated.

Keywords: attribute noise, random forest, software quality classification.

1 Introduction

The presence of noise in data is a recognized problem in software quality initiatives [11]. Noise can significantly affect the results and conclusions obtained from classification performance studies. For example, in the software quality estimation domain, models are built to distinguish program modules that are likely to contain faults (fault prone or fp) from those that are not fault prone (nfp). It is common in mission critical systems [8], that the misclassification of fp modules can threaten property and/or lives.

Relatively few studies, and none in software quality classification, have considered the impact of attribute noise on classification performance. A considerable number of studies in the data mining and machine learning field have focused solely on the impact of class noise on classification performance. In this study, we injected domain realistic attribute noise into seven class-imbalanced[1] real-world software engineering measurements datasets, initially relatively free of noise. The datasets were relatively cleansed of inherent noise before the injection of simulated attribute noise. Injecting noise into a dataset that already contains noise can bias any empirical conclusions. Thus, the experiments in this study were carefully designed to ensure result accuracy and robustness. The attributes selected for noise injection were ranked based on their KS statistic to determine their class prediction significance. Those with higher rankings were injected with domain realistic noise, in order of decreasing KS statistic [6].

The experimental results demonstrated that when the number of attributes injected with noise was five or more, or approximately 39% of the 13 independent attributes, the impact on classification performance became more important in this study. On the other hand, all classifiers had relatively minimal impact on their performances when one, two, three, or four of the most significant attributes were injected with noise. This performance behavior provides evidence that having moderate levels of attribute noise in software measurement datasets is not nearly as important to classification performance [11].

1.1 Related Work

Studies considering the impact of attribute noise on software quality classification have not been found. Often, related studies in other domains have suggested that in many cases, eliminating instances with class noise will improve classification accuracy [3, 7]. Further, very few of these studies have investigated the impact of attribute noise [18] on classification performance. Typically, handling attribute noise is more difficult than class noise [14, 18]. Class noise occurs when a program module is labeled as belonging to a class different than the one implied by its attributes, e.g., $fp \rightarrow nfp$ or when $nfp \rightarrow fp$. Often, eliminating instances which contain attribute noise is counter-productive, because other attributes belonging to the deleted instance can still contain valuable information. Zhu and Wu [18] pointed out an important fact from real-world data: the class information is usually much cleaner than commonly assumed; and it is the independent attributes that usually

*Readers may contact the authors through Taghi M. Khoshgoftaar, Empirical Software Engineering Laboratory, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA. Phone: (561)297-3994, Fax: (561)297-2800.

[1] In binary classification, if one of the two classes has more program modules, then the data is considered imbalanced relative to the class.

contain more noise. Our previous work [10] determined similar findings, corroborating the importance of measuring the impact of attribute noise on classification. In this study, we used 11 classifiers and two performance metrics to investigate the impact of attribute noise on software quality classification performance. No other related study has used this many classifiers to investigate the impact of attribute noise. In fact, to our knowledge, similar comprehensive experimental procedures to those used in this work have not been reported in any other related studies.

The remainder of this paper is organized as follows: Section 2 describes the 11 classifiers used. The experimental methodology is provided in Section 3. Experimental results are presented in Section 4, and the conclusion is provided in Section 5.

Table 1. Primitive Software Metrics

Line Count Metrics	$Total\_Lines\_of\_Code$
	$Executable\_LOC$
	$Comments\_LOC$
	$Blank\_LOC$
	$Code\_And\_Comments\_LOC$
Halstead Metrics	$Total\_Operators$
	$Total\_Operands$
	$Unique\_Operators$
	$Unique\_Operands$
McCabe Metrics	$Cyclomatic\_Complexity$
	$Essential\_Complexity$
	$Design\_Complexity$
Branch Count Metric	$Branch\_count$

2 Classifiers

Every classifier used was implemented in the Weka tool [17]. Default parameter changes were done only when classifier performance improved significantly. *C4.5* [13] is a benchmark decision tree learning algorithm. Two different versions of the C4.5 classifier were used. C4.5D uses the default parameter settings in Weka, while C4.5N uses no decision-tree pruning and Laplace smoothing [16]. The *random forest* (RF100) classifier [2] uses bagging and the 'random subspace method' to build an ensemble of randomized decision trees which are combined to produce the final prediction. RF100's 'Number of Trees' parameter was changed to 100 from its default value of 10. *K nearest neighbors* [1] (kNN) classifiers were built with changes to two parameters. The 'distanceWeighting' parameter was set to 'Weight by 1/distance'. Two different 'kNN' classifiers were built using $k = 2$ and $k = 5$ and were denoted '2NN' and '5NN', respectively.

The *support vector machine* (SVM) classifier called SMO in Weka had two changes to the default parameters: the complexity constant 'c' was set to 5.0 and 'buildLogisticModels' was set to 'true'. By default, a linear kernel was used. For a *Multilayer perceptrons* (MLP) classifier (a type of neural network), the 'hiddenLayers' parameter was changed to '3' to define a network with one hidden layer containing three nodes, and the 'validationSetSize' parameter was changed to '10' to cause the classifier to leave 10% of the training data aside to be used as a validation set to determine when to stop the iterative training process. *Radial basis function networks* (RBF) are another type of artificial neural network. The only parameter change for RBF was to set the parameter 'numClusters' to 10. *Naive Bayes* (NB) utilizes Bayes's rule of conditional probability and is termed 'naive' because it assumes conditional independence of the features. *Logistic regression* (LR) is a statistical regression model for categorical prediction. *RIPPER* (Repeated Incremental Pruning to Produce Error Reduction) is a rule-based classifier and is named JRip [4] in Weka. The default Weka parameters for these three classifiers were not changed.

3 Experimental Design

3.1 Experimental Datasets

The datasets used are from seven NASA software projects: JM1, CM1, MW1, PC1, KC1, KC2, and KC3 which were obtained from the NASA Metrics Data Program (MDP). Instances represent software modules with 21 software measurements. Classifiers were built using 13 primitive metrics[2] as independent variables and a module-class (binary) as the dependent variable, i.e., fp (fault-prone) or nfp (not fault-prone). The minority class is represented as the positive or fp class, while the majority class is represented as the negative or nfp class.

The RBCM noise filter was applied to these datasets in order to identify and remove subsets of noisy instances [9]. Table 2 provides details about the seven initial datasets and their respective cleansed versions. In this table, the #P column contains the number of positive (fp) examples, while the #N has the number of negative (nfp) examples. The %P column provides the percentage of positive examples relative to the total number of examples in a dataset, e.g., PC1 originally contained 1107 instances, of which 6.87% were fp. After cleansing, 703 total instances remained of which 7.54% were fp. The cleansed datasets were used in this work. These datasets were subjected to a methodical and carefully designed noise cleansing process described in [9] (also see Van Hulse [15] for a detailed discussion of the noise cleansing procedure).

Table 3 shows the classification performance across all classifiers obtained using each of the cleansed datasets. According to the AUC and KS performance metrics, the best classification performance was obtained by using JM1. The

[2] The other metrics are derived from the 13 primitive ones in Table 1

Table 2. Dataset characteristics

	Initial			Cleansed		
Data	# P	# N	% P	# P	# N	% P
JM1	470	2393	16.42	235	2210	9.61
CM1	48	457	9.50	39	277	12.34
MW1	31	372	7.69	20	291	6.43
PC1	76	1031	6.87	53	650	7.54
KC1	325	1782	15.42	271	1093	19.87
KC2	106	414	20.39	82	333	19.76
KC3	43	415	9.39	38	264	12.58

Table 3. Cleansed Datasets Performance

Data	AUC	Rank	Data	KS	Rank
JM1	0.9987	1	JM1	0.9974	1
KC1	0.9977	2	KC1	0.9763	2
KC2	0.9922	3	PC1	0.9607	3
PC1	0.9915	4	KC2	0.9532	4
KC3	0.9865	5	KC3	0.9521	5
CM1	0.9837	6	CM1	0.9487	6
MW1	0.9767	7	MW1	0.9428	7
Avg	0.9896		Avg	0.9616	

second best performance was obtained using KC1, and the worst performance was obtained using MW1. The average values ('Avg' row) across all the cleansed datasets and classifiers were used as a baseline reference for performance comparisons. Note that all datasets have nearly perfect performance by the classifiers, further supporting the fact that the datasets have been cleansed of noise significantly.

3.2 Performance Metrics

Traditional performance measures such as classification accuracy, or its complement, misclassification rate, are inappropriate when dealing with the classification of class imbalanced data. When as few as 1% of examples belong to the positive class, a classifier can achieve an accuracy of 99% by simply labeling all examples as belonging to the negative class. In a domain such as software quality classification, however, such a model is useless. Instead, two performance metrics that consider the ability of a classifier to differentiate between the two classes were used. The Kolmogorov-Smirnov statistic [6] measures the maximum difference between the empirical distribution function of the posterior probabilities of instances in each class.

The second metric used was the Receiver Operating Characteristic curve [12] (ROC). ROC curves graph true positive rates on the y-axis versus the false positive rates on the x-axis. The resulting curve illustrates the trade-off between detection and false alarm rates. Often, performance metrics consider only the default decision threshold of 0.5. ROC curves illustrate the performance across all decision thresholds. For a single numeric measure, the *area under the ROC curve* (AUC) is widely used, providing a general idea of the predictive potential of the classifier.

3.3 Noise Injection Procedure

The attributes selected for noise injection were identified by the KS test at a 5% significance level [9]. The KS two-sample test is a non-parametric statistical significance test [5]. It is useful in determining the fp and nfp discriminative quality of the attributes under consideration. The greater the KS statistic for an attribute, the better is its discriminative quality for segregating the fp instances from nfp instances. Noise was injected at five levels - 10%, 20%, 30%, 40%, and 50%. Noise was first injected into the most significant predictive attribute, creating a total of 35 derived datasets from the seven cleansed datasets. The derived datasets from each cleansed dataset contain a different amount of corruption in that attribute. The next 35 datasets were obtained by corrupting both the most and second most significant attributes. This procedure was repeated until the seven most significant attributes were corrupted. The results of six attributes injected with noise were excluded from this study because of similarities to the results obtained when seven attributes were used. A noise level of 10% implied that the values for the selected attributes were corrupted for 10% of the instances. The corruption was obtained by replacing the given value with a randomly selected value reflecting an instance of the opposite class, i.e., nfp or fp. For a given injected noise level, the nfp and fp proportions of the instances injected with noise was approximately the same as the nfp and fp proportions of the given dataset. For example, if the given dataset had a proportion of 70:30 for nfp:fp instances and if 100 instances were injected with noise, then those 100 instances would consist of 70 nfp and 30 fp instances.

3.4 Experimental Design Summary

The models were trained using the derived datasets from the cleansed datasets. 10-fold cross validation was used to build and test the models. The datasets were broken into 10 partitions, where nine of the 10 partitions were used to train the model, and the remaining (hold out) partition was used to test the model. This was repeated 10 times so that each partition was used as hold out data once. In addition, 10 independent repetitions of each experiment were done to avoid any bias that may occur during the random selection process and to ensure the statistical significance of the results. The results reported in the following sections represent the average of these repetitions. Noise was injected in up to seven of the most significant attributes from each dataset. There were five levels of noise, 10%, 20%, 30%,

Table 4. Overall Impact of Attribute Noise

	Clean	1-attr	2-attr	3-attr	4-attr	5-attr	7-attr
AUC	0.9896	0.9875	0.9856	0.9812	0.9794	0.9651	0.9504
KS	0.9616	0.9506	0.9429	0.9310	0.9268	0.8987	0.8542

Table 5. Noise Impact on Classifiers by AUC

	1-attr	2-attr	3-attr	4-attr	5-attr	7-attr
C4.5N	0.9826	0.9852	0.9815	0.9811	0.9667	0.9621
NB	0.9939	0.9919	0.9879	0.9833	0.9721	0.9499
MLP	0.9875	0.9875	0.9865	0.9850	0.9786	0.9456
RIPP	0.9764	0.9674	0.9547	0.9543	0.9367	0.9088
5NN	0.9946	0.9924	0.9887	0.9863	0.9816	0.9721
SVM	0.9941	0.9906	0.9941	0.9933	**0.9647**	**0.9700**
RF100	**0.9989**	**0.9991**	**0.9973**	**0.9972**	**0.9907**	**0.9898**
RBF	0.9661	0.9622	0.9618	0.9574	0.9438	0.9331
LR	0.9932	0.9929	0.9896	0.9925	0.9762	0.9662
C4.5D	0.9827	0.9832	0.9698	0.9666	0.9337	0.8956
2NN	0.9925	0.9890	0.9815	0.9768	0.9709	0.9608

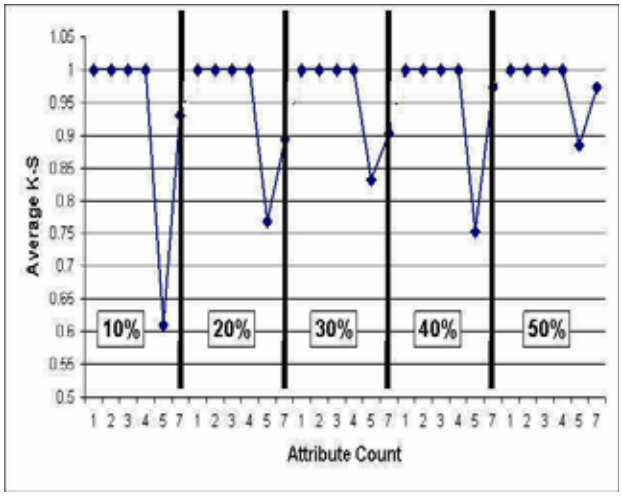


Figure 1. SVM Performance using JM1

40%, and 50%, eventually injected into these attributes. Based on these experimental parameters, there were a total of 231,000 models built and evaluated.

4 Experimental Results

4.1 Noise Impact on Classification Performance

Table 4 contains the impact of attribute noise on classification performance averaged overall datasets and learners. The column labeled 'Clean' contains the performance values obtained when using the cleansed datasets. The column labeled '1-attr' contains the performance values overall learners and datasets when the most significant attribute was injected with noise. The rest of the columns (labeled '2-attr,' '3-attr,' '4-attr,' etc.) show the performance values when noise was injected into two, three, four, five, and seven of the most significant attributes. Both metrics showed that when there were five attributes injected with noise (in the column labeled '5-attr') a relatively large drop in classification performance occurred. If we compare the performance values from the 'Clean' column to the rest of the performance values from the other columns, we can clearly see the relative large performance decline in column '5-attr'. Furthermore, the largest performance losses were 4.0% for the AUC and 10.7% for the KS metric. These losses were obtained by calculating the percent difference between the 'Clean' baseline and the '7-attr' column values.

Tables 5 and 6 contain the impact on each classifier's performance overall datasets and noise levels as the number of attributes injected with noise increased. The best performing classifier (RF100) is bolded. Both tables show declining classifier performance as the number of attributes with injected noise is increased. Note that the right most column in these tables (showing seven corrupted attributes and labeled '7-attr') has the lowest classification performance. The only exception to this trend in both tables is observed in Table 5. In this table, SVM is bolded to show its performance improvement when noise injection increased from five to seven attributes. SVM's unusual performance using JM1 can be used as an example of a classifier with unexpected results in a particular case study and it is further explored in Figure 1. Note that all classifiers maintained very good performances when one, two, three, or four of the most significant attributes were injected with noise.

Figure 1 shows SVM's performance based on the average KS metric (in the y-axis) when using the derived datasets from the best performing dataset, JM1 (AUC results are similar but not shown due to space limitations). The x-axis has the number of attributes injected with noise per injected noise level. At the lowest level of injected noise, 10% (labeled at the bottom of the figure), SVM had its largest performance loss when noise injection was increased from four to five attributes. In contrast, SVM had a relatively large performance improvement when noise injection was further increased from five to seven attributes. The classification performance was nearly perfect for the first four attributes with injected noise, regardless of the level of noise. In comparison to the other noise levels (separated by the solid vertical lines in Figure 1), 20%, 30%, 40%, and 50%, SVM recorded its best performance at the highest level of noise, 50%. This unusual SVM performance can be partly explained as an anomaly in the performance of this classifier when using the derived datasets from JM1. Under no other experimental scenario did SVM's (nor any other classifier's) performance resem-

Table 6. Noise Impact on Classifiers by KS

	1-attr	2-attr	3-attr	4-attr	5-attr	7-attr
C4.5N	0.9620	0.9544	0.9422	0.9389	0.9026	0.8734
NB	0.9485	0.9372	0.9181	0.9036	0.8767	0.8117
MLP	0.9205	0.9195	0.9178	0.9092	0.8962	0.8106
RIPP	0.9527	0.9294	0.9033	0.9042	0.8877	0.8147
5NN	0.9482	0.9436	0.9404	0.9371	0.9245	0.8899
SVM	0.9654	0.9593	0.9541	0.9463	0.8977	0.8629
RF100	**0.9860**	**0.9854**	**0.9746**	**0.9726**	**0.9500**	**0.9324**
RBF	0.9241	0.9184	0.9135	0.9068	0.8778	0.8495
LR	0.9531	0.9521	0.9426	0.9492	0.9122	0.8714
C4.5D	0.9599	0.9495	0.9253	0.9202	0.8666	0.8134
2NN	0.9357	0.9228	0.9096	0.9064	0.8937	0.8669

Table 7. Overall Performance of Classifiers

	AUC_c	AUC_n	R_n	KS_c	KS_n	R_n
RF100	0.9997	**0.9955**	**1**	0.9912	**0.9668**	**1**
5NN	0.9974	0.9859	2	0.9623	0.9306	3
LR	0.9947	0.9851	3	0.9624	0.9301	4
SVM	0.9980	0.9845	4	0.9807	0.9309	2
NB	0.9959	0.9798	5	0.9619	0.8993	8
2NN	0.9948	0.9786	6	0.9535	0.9058	6
MLP	0.9908	0.9784	7	0.9404	0.8987	9
C4.5N	0.9824	0.9765	8	0.9661	0.9289	5
C4.5D	0.9811	0.9553	9	0.9619	0.9058	7
RBF	0.9767	0.9541	10	0.9444	0.8983	10
RIPP	0.9736	0.9497	11	0.9526	0.8956	11

ble the one obtained using JM1's derived datasets.

4.2 Noise Impact on Classifiers

In this section we investigate the impact of attribute noise on each of the 11 classifiers averaged over all datasets, all noise levels and all attributes injected with noise. The impact on performance using the JM1 and MW1 datasets is also presented. Note that there are small disagreements between the AUC and KS values in Tables 7, 8, and 9. In general, the top two or three performing classifiers have very close values and rankings. This observation is also true for the two lowest performing classifiers. The tables have the same column format showing (from left to right) the classifier name, the AUC_c value obtained from the cleansed datasets, the AUC_n value obtained from the derived datasets with injected noise, the derived ranking (R_n) based on AUC_n, the KS_c value obtained from the cleansed datasets, the KS_n value obtained from the derived datasets with injected noise, and the corresponding ranking (R_n) from KS_n. The best ranked classifier (RF100) is bolded in the tables which are ordered by AUC_n ranking.

Table 7 shows the impact of noise by the lower AUC_n values when compared to those values from AUC_c. The same trend is observed in the KS values. The rankings are based on the AUC_n and KS_n values respectively. The best performance was obtained by the RF100 classifier, regardless of the levels of noise, number of significant attributes with injected noise, dataset used, or metric used. The second best performing classifiers were 5NN (as ranked by the AUC_n) and SVM (by the KS_n). The classifiers most affected by attribute noise were RBF and RIPPER.

Table 8. Performance of Classifiers using JM1

	AUC_c	AUC_n	R_n	KS_c	KS_n	R_n
RF100	1.0	**0.9991**	**1**	1.0	**0.9962**	**1**
5NN	1.0	0.9988	2	1.0	0.9941	3
2NN	1.0	0.9986	3	1.0	0.9926	4
LR	1.0	0.9985	4	0.9999	0.9950	2
MLP	1.0	0.9981	5	1.0	0.9897	5
RBF	0.9998	0.9956	6	1.0	0.9889	7
NB	0.9992	0.9955	7	0.9985	0.9846	10
C4.5N	0.9958	0.9949	8	0.9915	0.9893	6
C4.5D	0.9957	0.9919	9	0.9915	0.9879	8
RIPP	0.9951	0.9919	10	0.9906	0.9849	9
SVM	1.0	0.9682	11	1.0	0.9506	11

Table 9. Performance of Classifiers using MW1

	AUC_c	AUC_n	R_n	KS_c	KS_n	R_n
RF100	0.9991	**0.9814**	**1**	0.9859	**0.9113**	**1**
SVM	0.9998	0.9758	2	0.9959	0.8952	2
5NN	0.9933	0.9747	3	0.9297	0.8923	3
NB	0.9942	0.9698	4	0.9564	0.8485	7
MLP	0.9991	0.9693	5	0.9883	0.8894	4
LR	0.9902	0.9674	6	0.9470	0.8834	6
2NN	0.9787	0.9595	7	0.8897	0.8464	8
C4.5N	0.9494	0.9550	8	0.9359	0.8865	5
C4.5D	0.9508	0.9215	9	0.9366	0.8315	9
RIPP	0.9370	0.9010	10	0.8966	0.8132	10
RBF	0.9519	0.8907	11	0.9085	0.7995	11

Tables 8 and 9 show the performance of the classifiers when using the derived datasets from JM1 and MW1 (see Table 3 for datasets details). Both tables show the impact of attribute noise by the lower AUC_n values when compared to those values from AUC_c. The same trend is observed in the KS values. The rankings are based on the AUC_n and KS_n values respectively.

Table 8 shows RF100 as the best performing classifier, followed by 5NN (for AUC_n) and LR (for KS_n). The classifier most affected by noise was SVM. Recall that both SVM and 5NN had the second best overall performance in this study. We explained SVM's performance when using JM1 in the previous section with a particular emphasis on the line plot shown in Figure 1. On the other hand, Table 9 shows RF100 as the best performing classifier, with SVM and 5NN as the second and third best classifiers re-

spectively. RBF and RIPPER had the lowest classification performance. These results agreed with the performances presented in Table 7 and were very similar to the results observed from KC1, KC2, KC3, PC1, and CM1. The results from these datasets were not presented due to space limitations.

5 Conclusions

The quality of the software measurement data is of paramount importance to software quality classification. As the levels of attribute noise increased, every classifier's performance decreased. This behavior was observed by the decline in the recorded AUC and KS values. Furthermore, when the number of significant attributes injected with noise increased, the classifiers' performances decreased. A threshold for this number of attributes was identified that when reached or exceeded, it drastically affected classification performance. The threshold value was determined to be five attributes, or approximately 39% of the 13 attributes in the datasets. In contrast, all classifiers had minimal impact on their classification performance when one, two, three, or four of the most significant attributes were injected with noise. This performance behavior implies that moderate attribute noise is not as concerning to classification performance as class noise. To our knowledge, this is the first software quality classification study to report such important empirical results.

The AUC and KS values were used to determine the best and worst performing classifiers. The empirical results conclusively demonstrated that the random forest ensemble classifier obtained the best and most consistent classification performance in every experimental scenario. The second best classifiers were SVM and 5NN. The classifiers most affected by attribute noise were RBF and RIPP. Based on these results, we strongly recommend the random forest classifier for software quality estimation. Even though SVM had very good overall classification performance, when the JM1 dataset was used with SVM, we observed the worst classification performance. At the lowest level of injected noise, 10%, SVM had its largest performance loss. This loss was observed in the KS and AUC values when noise injection was increased from four to five attributes. On the other hand, SVM had a relatively large performance improvement when noise injection was further increased from five to seven attributes. In comparison to the rest of the noise levels, 20%, 30%, 40%, 50%, SVM recorded its best performance at 50% (see Figure 1). These results emphasize the importance of using high quality data and carefully designed comprehensive experimental procedures.

Future work will include the investigation and comparison of the results obtained from the injection of class noise. Further research will consider both attribute and class noise, and will include additional software measurement datasets.

References

[1] D. W. Aha. *Lazy learning*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[2] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[3] C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[4] W. W. Cohen. Fast effective rule induction. In *Proc. 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[5] W. Conover. *Practical Nonparametric Statistics*. John Wiley and Sons, NY, 1971.

[6] D. J. Hand. Good practice in retail credit scorecard assessment. *Journal of the Operational Research Society*, 56:1109–1117, 2005.

[7] A. Karmaker and S. Kwek. A boosting approach to remove class label noise. *5th International Conference on Hybrid Intelligent Systems*, pages 206–211, 2005.

[8] T. M. Khoshgoftaar, E. B. Allen, and J. Deng. Using regression trees to classify fault-prone software modules. *IEEE Trans. Reliability*, 51(4):455–462, 2002.

[9] T. M. Khoshgoftaar, N. Seliya, and K. Gao. Detecting noisy instances with the rule-based classification model. *Intelligent Data Analysis: An International Journal*, 9:347–364, 2005.

[10] T. M. Khoshgoftaar and J. Van Hulse. Empirical case studies in attribute noise detection. In *Proceedings of the IEEE International Conference on Information Reuse and Integration*, pages 211–216, Las Vegas, NV, August 2005.

[11] T. M. Khoshgoftaar, S. Zhong, and V. Joshi. Enhancing software quality estimation using ensemble-classifier based noise filtering. *Intelligent Data Analysis: An International Journal*, 6(1):3–27, 2005.

[12] F. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42:203–231, 2001.

[13] J. R. Quinlan. *C4.5: Programs For Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.

[14] C. M. Teng. Correcting noisy data. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 239–248, 1999.

[15] J. Van Hulse. Data quality in data mining and machine learning. *Ph.D. Dissertation, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL USA*, May 2007. Advised by T. Khoshgoftaar.

[16] G. M. Weiss and F. Provost. Learning when training data are costly: the effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19:315–354, 2003.

[17] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, California, 2nd edition, 2005.

[18] X. Zhu and X. Wu. Class noise vs. attribute noise: A quantitative study of their impacts. *Artificial Intelligence Review*, 22:177–210, 2004.

An Adaptive Neural Network with Dynamic Structure for Software Defect Prediction

Zhiwei Xu and Naeem Seliya
Computer and Information Science
University of Michigan – Dearborn
4901 Evergreen Road
Dearborn, MI 48128 USA
{zwxu,nseliya}@umich.edu

Weibiao Wu
Department of Statistics
University of Chicago
5734 S. University Avenue
Chicago, IL 60637 USA
wbwu@galton.uchicago.edu

Abstract

A timely software defect prediction activity is a useful tool in improving the quality and reliability of software-based systems. Such a model provides the quality improvement team with a guidance on which program modules, prior to system testing and deployment, should be allocated the limited re-inspection and quality improvement resources. Among existing software fault prediction techniques, artificial neural networks are effective in extracting the complex relationship between the different software measurements of program modules and their associated defect data. We present an innovative neural network model that alleviate the various problems associated with the traditional back-propagation neural network models. The proposed Adaptive Resilient Propagation Neural Network (APROP) model for software defect prediction is investigated with a case study of a real-world high assurance software system. The APROP defect prediction model is compared with the traditional back-propagation neural network, and those results indicated a significant improvement in fault prediction. A comparison with a non-neural network model, Multiple Linear Regression, further validated the improved fault prediction of the proposed model. The APROP model is not limited to software defect prediction, and can be applied for other estimation problems during software development, such as identifying critical test cases for regression testing.

1. Introduction

Assuring good software quality and reliability is essential to any software development project, especially for high assurance and mission-critical software systems. In software engineering practice, various techniques and processes are used improving the quality and reliability of software products. Some of those techniques include software inspection [6], software redesign [11], formal requirements analysis [16], and knowledge-based software quality models [3, 8, 15]. This paper focuses on the latter approach to software quality improvement.

It is known that the presence of software errors and faults has a direct impact on the achieved level of quality and reliability of software-based systems. Hence, reducing the presence of software faults in program modules is a practical way to assure an improved level of software quality. Knowledge-based software engineering includes quantitative software quality estimation modeling and analysis. A software quality prediction model is typically built by training a machine learner (prediction model) on known software measurements and defect data collected from a prior release or previously developed system. Upon validation, the trained software quality model can be applied to predict the unknown defect data for program modules of the currently under-development project.

The characterization of defect data (software quality) varies from project to project, and depends on goals of the software quality improvement team. In some cases, software quality is expressed and evaluated in terms of membership of program modules to different quality-based groups, such as high-quality and low-quality. In other instances, software quality is expressed and evaluated in terms of number of faults associated with different program modules. In some other analysis, the amount of code churn required to fix a reported problem is used to express and evaluate software quality. We focus on number of faults in a program module as the primary characteristic of software quality.

We present an innovative artificial neural network model that overcomes several shortcomings of traditional neural networks, and investigate its performance and viability as a software quality prediction model. To demonstrate its bet-

ter performance for software fault prediction, the proposed Adaptive Resilient Propagation Neural Network (denoted as APROP) is compared with the traditional steepest gradient artificial neural network (denoted as ANN). We also compare the APROP model with a non-neural network software quality model; and for that purpose, we use the Multiple Linear Regression (denoted as MLR) prediction method.

The APROP neural network model is investigated with a case study of software measurement and defect data obtained from a real-world military command, control, and communication system. We build three software fault prediction models, i.e., based on the APROP, ANN, and MLR models. The three models are compared for statistical significance in their relative prediction performances. It is shown that the proposed Adaptive Resilient Propagation Neural Network model provides significantly better fault predictions compared to both ANN and MLR models.

While other algorithms and models have been used for software fault prediction, the focus of this paper is limited to presenting the APROP model as a viable and practical alternative to the traditional back-propagation neural network. We also include MLR in our comparison with APROP; however, a comprehensive comparison with other prediction algorithms and models is out of scope for this paper – in part due to paper size considerations. The application of our APROP model is not limited to fault prediction, and can be extended to predict other attributes in software project development, e.g., identification of critical test cases for regression testing.

The remainder of the paper is structured as follows: Section 2 summarizes key related work in the literature; Section 3 presents the Adaptive Resilient Propagation Neural Network model; Section 4 summarizes Multiple Linear Regression which we compare with the APROP model; Section 5 describes our case study, and analyzes the obtained empirical results; and, Section 6 concludes our paper and includes suggestions for future research directions.

2. Related Work

A perfect software development environment would include software quality improvement activities being applied to all program modules. However, limited resources available for re-inspections and quality improvement require a targeted allocation of those resources for maximum software quality improvement. The output of software quality estimation models are predictions that a development team can utilize to identify high-risk or low-quality program modules in the system. Knowledge-based software quality estimation models generally include software quality classification models [5, 9], software fault prediction models [2], and module-order models [7].

A software quality classification model aims to maximize return on quality improvement activities by guiding in predicting which program modules are fault-prone or not-fault-prone. A software fault prediction model estimates the number of faults a given module for use as a guide to target available resources to program modules that are likely to have more faults. Finally, a module-order software quality model provides the quality improvement team with a quality-based ranking of program modules which can then be used for targeted software quality improvement efforts.

In the literature, one can find various methods and techniques that have been investigated for building software quality estimation models, including decision trees [9], regression [15], case-based reasoning [3], soft computing methods [7, 18], artificial neural networks [17], etc. This paper does not include an extensive coverage of various software quality prediction methods, as it is beyond its scope. However, readers are provided with sufficient background material in the cited references.

3. Adaptive Resilient Propagation Network

Artificial neural networks (ANN) are systems that are deliberately constructed to make use of some organizational principles resembling those of the human brain. ANN have been studied for a long time since Rosenblatt [13] first introduced single layer perceptrons. For multilayer neural networks, back-propagation [10, 14, 19] is the most popular training algorithm.

We designed an adaptive resilient propagation neural network (APROP) based on the resilient propagation, adaptive learning rate, momentum, and structure optimization technique to overcome the inherent disadvantages of slow training speed, large memory usage, potential diverge training problems, and local minimum associated with back-propagation algorithms.

A typical neural network has the input layer, a numbers of hidden layers, and the output layer. Figure 1 shows a feedforward multilayer neural network. The traditional back-propagation algorithm adjusts the weights and basis by application of a gradient to compute the influence of each weight in the network with respect to a cost function E.

$$\frac{\partial E}{\partial w_{ij}} = -\sum_{k=1}^{p}(d_i^{(k)} - o_i^{(k)})f'(net_i^{(k)})x_j^{(k)}$$

However, training a multilayer network is usually a time-consuming process, and it is not easy to achieve the global optimal solution. Many algorithms have been proposed to speed up the learning process and achieve better solutions. For example, Martin Riedmiller and Heinrich Braun [12] proposed adapting the partial derivative of weights to optimize the learning process. APROP differs from traditional

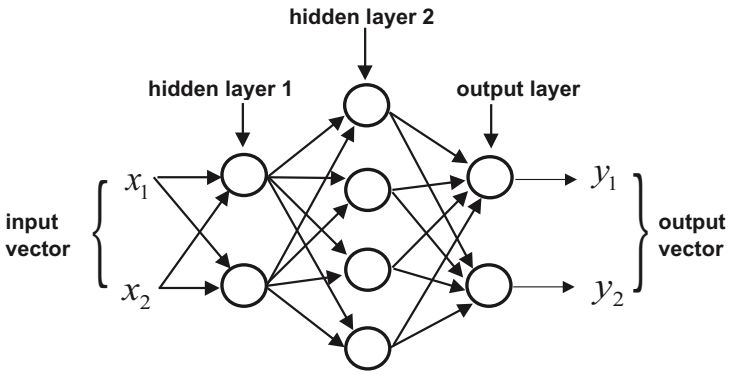


Figure 1. A feedforward neural network

resilient propagation networks in that both its learning rate and changes in weight derivatives are adaptive, and a growing method is used to automatically search for the optimized structure. Making both learning rate and the weight derivatives adaptive, and optimizing structure of networks can achieve better prediction and a faster and more stable learning process.

In summary, the ARPROP learning algorithm can be outlined in the following way. Consider a network with H layers, $h = 1, 2, ..., H$, and let net_i^h and o_i^h denote the net input and output of the ith unit in the h layer, respectively and w_{ij}^h denote the connection weight from o_j^{h-1} to o_i^h. Suppose the network has m input nodes and n output nodes. There are p pairs of training pairs $\{(\mathbf{x}^{(k)}, \mathbf{d}^{(k)})\}$ where $k = 1, 2, ..., p$.

1. Initialization: Choose a learning rate r, maximum epoch number and training goal E_{goal}. Randomly select weights and basis for each neuron.

2. Training loop: Apply the kth input pattern to the input layer $(h = 1)$.

3. While not done: the whole set of training data has been cycled through once.

   (a) Forward propagation: Propagate the signal forward through the network using

   $$o_i^h = f(net_i^h) = f\left(\sum_j w_{ij}^h \, o_j^{h-1}\right) \tag{1}$$

   for each i and h until the outputs of the output layer o_i^h have all been obtained.

   (b) Output error measurement: Compute the error value and error signals ∇E_i^h for the output layer:

   $$\begin{aligned} E &= \frac{1}{2}\sum_{i=1}^{n}(d_i^{(k)} - o_i^h)^2 \\ \frac{\partial E}{\partial w_{ij}} &= -\sum_{k=1}^{p}(d_i^{(k)} - o_i^{(k)}) f'(net_i^{(k)}) x_j^{(k)} \\ \nabla E_i^h &= r * \frac{\partial E}{\partial w_{ij}} \end{aligned}$$

   (c) Error back-propagation: Propagate the errors backward to update the weights and compute the error signals ∇E_i^h for the preceding layers:
   For all weights and biases

   $$\begin{aligned} \nabla w_{ij}^h(n+1) &= f_{adapt}(\frac{\partial E}{\partial w_{ij}}(n), \nabla w_{ij}^h(n), \\ &\qquad \frac{\partial E}{\partial w_{ij}}(n+1), r) \\ w_{ij}^h &= w_{ij}^h + \nabla w_{ij}^h(n+1) \end{aligned}$$

   where f_{adapt} is an adaptive function.

   (d)

   $$\nabla w_{ij}^{h-1} = f'(net_i^{h-1} \sum_j w_{ij}^h \, \nabla E_i^h)$$

   where $h = H, H-1, ..., 2$.

   (e) update learning rate according to $r = \frac{0.01}{1+e^{-1.4*E}}$

4. One epoch iteration: End while

5. Stop training check: If the total error reach the training goal E_{goal} or the epoch reach the set maximum, stop training, otherwise go back to the training loop for the next epoch.

The above APROP is for a fixed-node hidden layer ANN. If the hidden layer units are few, however, the ANN's learning ability is reduced, and the ANN may not converge and can not remember any learning pattern. On the other hand, if the hidden layer units are too many, the unnecessarily complex network can easily result in over-fitting of the software quality model. Hence, we empirically tried to find an optimal number of hidden layer units for the ANN by changing the number of nodes in hidden layer until the best prediction quality was achieved.

Initially, the number of hidden layer units was made equal to that of input nodes and then the number of nodes in hidden layer was changed until the best prediction quality was achieved. The input data was randomly divided into two subsets: (1) training data used to train the neural networks, and (2) validating data used to test the training quality. The structure of the neural network that had the best training result was chosen.

4. Multiple Linear Regression

Multiple linear regression is a statistical method for estimating a dependent variable as a function of independent variables [1]. The model is based as an equation where the dependent variable (number of faults in our study) is expressed in terms of predictors (software metrics in our study), and is generally given by,

$$\hat{y}_i = a_0 + a_1 x_{i1} + \cdots + a_p x_{ip}$$
$$y_i = a_0 + a_1 x_{i1} + \cdots + a_p x_{ip} + e_i$$

where, $x_{i1}, \ldots, x_{ip}$ are the independent variables values, $a_0, \ldots, a_p$ are the parameters to be estimated, $\hat{y}_i$ is the dependent variable to be predicted, y_i is the actual value of the dependent variable, and $e_i = y_i - \hat{y}_i$ is the error in prediction for the ith case.

The software measurement data is initially subjected to statistical analysis for removing any correlation existing between independent variables and to remove insignificant independent variables. The process of determining the independent variables which are significant is known as model selection, and several methods exist. They are forward elimination, stepwise selection and backward elimination. Here, stepwise regression is used. Stepwise regression [1] selects an optimal set of independent variables for the model. In this process, variables are either added or deleted from the regression model at each step of the model building process. Once the model is selected, the parameters $a_0, \ldots, a_p$ are then estimated using the least squares method.

Table 1. Software Product Metrics

Symbol	Description
η_1	Number of unique operators.
N_1	Total number of operators.
η_2	Number of unique operands.
N_2	Total number of operands.
$V(G)$	McCabe's cyclomatic complexity.
N_L	Number of logical operators.
V	$N \log_2(\eta_1 + \eta_2)$ is Halstead volume
$\widehat{N}$	$\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$ is Halstead length.
LOC	Lines of code.
ELOC	Executable lines of code.
PROCS	Number of procedures in a package.
COM	Number of command lines.
BLNK	Number of blank lines.

5. Software Project Case Study

5.1. System Description

We studied a large military command, control and communication system implemented in Ada. The software system was developed in a large organization by professionals using the procedural programming paradigm. The fault/defect data from problem tracking reports generated during the system integration and test phase. The defect and software measurement data is aggregated at the module level, where a program function or method is considered a program module. The number of faults (Y) in a program module is the dependent variable in our software quality research, and is predicted by a set of software metrics collected for the system.

The software metrics used in our case study are shown in Table 1, and include Halstead's, McCabe's and Statement-related product metrics [4]. The type and number of metrics used for the case study system were primarily governed by their availability, internal workings of the project, and the data collection tools used. Other metrics, including software process were not available. The use of the specific software metrics in our study does not advocate their effectiveness; hence, a different project may consider a different set of software measurements for analysis [3].

The case study data consisted of all 282 program modules measured by the development team. To create an independent evaluation dataset, we apply data splitting and randomly partition the original dataset into two subsets. The fit or training dataset consisted of two thirds of the program modules, while the remaining one-third formed the test (evaluation) dataset. Using an independent set for model evaluation provides unbiased software quality prediction results. Table 2 provides some summary statistics

Table 2. Statistics for number of faults

Statistic	Fit Data	Test Data
# Modules	188	94
Min.	0	0
Max.	29	42
Mean	2.27	2.56
Std. Dev.	4.65	5.88

on number of faults for the program modules in the fit and test datasets.

5.2. Performance Metrics

The performance accuracy of a given fault prediction model is evaluated with respect to the following metrics computed for the test dataset: *Average Absolute Error* (AAE), *Average Relative Error* (ARE), and the percentage of estimates that are within 20% (PRED(20)) and 25% (PRED(25)) of the actual number of faults value. With respect to the AAE and ARE performance metrics, lower values indicate better fault prediction accuracy. With respect to PRED(20) (or PRED(25)), a model with perfect fault prediction ability would have a PRED(20) (or PRED(25)) of 100% implying that it would estimate within 20% (or 25%) of the actual number of faults, 100% of the time.

The AAE and ARE metrics are given by, AAE $= \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y}_i|$, and ARE $= \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \widehat{y}_i}{y_i+1}\right|$, where n is the number of modules in the test data, y_i and $\widehat{y}_i$ represent the actual and predicted number of faults, respectively. In the case of ARE, since the actual number of faults may be zero, we add a "1" to the denominator to avoid division by zero.

5.3. Empirical Results and Discussion

We applied the multiple linear regression model (MLR), APROP, traditional steepest gradient neural network (ANN) on the case study data, respectively. The multiple linear regression model in this section predicted number of defects in the software modules. These predictions were based upon a selection of principal components after conducting principal components analysis for 95% variance. The first step of model selection was followed by fitting the model, and finally analyzing of the quality of fit and prediction accuracy of the software fault prediction model.

Stepwise regression selected $V(G)$, N_L, η_1, η_2 at the 5% significance level. The following model was obtained using the least-squares estimation technique.

$$Y = 0.054 + 0.0306\,\eta_2 - 0.0318\,V(G) + 0.2310\,N_L - 0.0425\,\eta_1$$

Table 3. Prediction Accuracy for Models

Models	AAE	ARE	PRED(20)	PRED(25)
MLR	1.787	0.700	57.4%	57.4%
ANN	2.064	0.710	60.0%	60.6%
APROP	1.159	0.513	70.2%	71.3%

Each variable (software metric) in the model was significant at $\alpha < 0.04$. The quality of fit of the trained software quality model was indicated by an $R^2 = 0.738$. Application of the model to the test dataset yielded an average absolute error of 1.787, i.e., AAE = 1.787.

In order to be consistent with the multiple linear regression, the neural network model used the same independent variables ($V(G)$, N_L, η_1, η_2) as the multiple linear regression model. We built two neural networks. One is APROP and the other is a traditional steepest gradient neural network (ANN). We compared prediction performances of the models based on the AAE, ARE, PRED(20), and PRED(25) values computed for the test dataset. Table 3 shows that APROP performed better than the other two methods.

Toward verifying the improved performance of APROP against MLR, we conducted a paired t-test with AAE as the response variable. The hypothesis test is formulated as, $\mu(Y_{MLR})$

$$H_0 : \mu(Y_{MLR}(\mathbf{x}_i) - Y_{APROP}(\mathbf{x}_i)) = 0$$
$$H_A : \mu(Y_{MLR}(\mathbf{x}_i) - Y_{APROP}(\mathbf{x}_i)) <> 0$$

A 95% confidence interval of {0.065, 1.190} for the mean difference between MLR and APROP doesn't include a zero, suggesting a significant difference between them. An observed $t = 2.21$, which is greater than the critical value $t_{1-\alpha\,;\,n-1}$, where $\alpha = 0.05$ and $n = 94$ (the number of program modules in the test dataset) in this case study – hence, $t_{0.05\,;\,93} = 1.66$. The *p*-value for this test is 0.029. The small p-value further suggests that the data are inconsistent with $H_0 : \mu(d) = 0$, that is, the two AAEs are not close to zero. Therefore, we reject the null hypothesis H_0, and concluded that APROP's AAE is significantly lower than MLR's in this case study.

We also conducted a paired t-test, with AAE as the response variable, to evaluate the significance of APROP's fault prediction accuracy compared to that of ANN. The hypothesis test is once again formulated as,

$$H_0 : \mu(Y_{ANN}(\mathbf{x}_i) - Y_{APROP}(\mathbf{x}_i)) = 0$$
$$H_A : \mu(Y_{ANN}(\mathbf{x}_i) - Y_{APROP}(\mathbf{x}_i)) <> 0$$

The 95% confidence interval for mean difference between ANN and APROP is $(0.046, 1.763)$, which doesn't include a zero and $t = 2.09$ is larger than critical value. Hence, we reject the null hypothesis H_0, and conclude that the

APROP model significantly reduces the AAE metric in this case study.

6. Conclusion

A software defect prediction model can aid the software quality improvement team in conducting a more focused quality improvement by identifying program modules likely to have more defects. We presented the Adaptive Resilient Propagation Neural Network as a valuable alternative to the traditional back-propagation neural network model for software defect prediction. The proposed APROP model is shown to yield statistically significant performance improvement compared to both the traditional back-propagation neural network model and the multiple linear regression model.

The black-box characteristic of artificial neural network models make them less attractive to analysts, including software practitioners. Hence, one has to consider both model-comprehension and model-accuracy when determining which defect prediction model to use. When it comes to absolute reduction of latent software faults, accuracy of a defect prediction model should be more important. When it comes to assessing the intricacies of the software development process, a white-box software defect prediction model is more attractive. As there is no one-solution-fits-all for software defect prediction, the analyst should consider various models (including APROP) and decide which one best suits their project's quality improvement goals.

Future work will include: additional case studies with other real-world software projects toward further validation of the benefits obtained by the APROP model; developing ways to further improve the architecture and performance of the APROP model; and, extending the APROP model for other software engineering prediction problems.

References

[1] M. L. Berenson, D. M. Levine, and M. Goldstein. *Intermediate Statistical Methods and Applications: A Computer Package Approach*. Prentice Hall, Englewood Cliffs, NJ, USA, 1983.

[2] V. U. B. Challagulla, F. B. Bastani, Y. I-Ling, and R. A. Paul. Empirical assessment of machine learning based software defect prediction techniques. In *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 263–270. IEEE Computer Society, February 2005.

[3] K. E. Emam, S. Benlarbi, N. Goel, and S. N. Rai. Comparing case-based reasoning classifiers for predicting high-risk software componenets. *Journal of Systems and Software*, 55(3):301–320, 2001. Elsevier Science Publishing.

[4] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Company: ITP, Boston, MA, 2nd edition, 1997.

[5] L. Guo, B. Cukic, and H. Singh. Predicting fault prone modules by the dempster-shafer belief networks. In *Proceedings of the 18th International Conference on Automated Software Engineering*, pages 249–252, Montreal, Quebec, Canada, October 2003. IEEE Computer Society.

[6] M. Kalinowski and G. H. Travassos. A computational framework for supporting software inspections. In *Proceedings of 19th International Conference on Automated Software Engineering*, pages 46–55. IEEE Computer Society, September 2004.

[7] T. M. Khoshgoftaar, Y. Liu, and N. Seliya. Module-order modeling using an evolutionary multi-objective optimization approach. In *Proceedings of 10th International Software Metrics Symposium*, pages 159–169, Chicago, IL, September 2004. IEEE Computer Society.

[8] T. M. Khoshgoftaar and N. Seliya. Fault prediction modeling for software quality estimation: Comparing commonly used techniques. *Empirical Software Engineering Journal*, 8(3):255–283, September 2003.

[9] T. M. Khoshgoftaar and N. Seliya. Comparative assessment of software quality classification techniques: An empirical case study. *Empirical Software Engineering Journal*, 9(3):229–257, 2004.

[10] Y. LeCun. A learning procedure for asymmetric network. *Cognitiva*, 85:599–604, 1985.

[11] G. Masuda, N. Sakamoto, and K. Ushijima. Redesigning of an existing software using design patterns. In *Proceedings of the International Symposium on Principles of Software Evolution*, pages 165 – 169, Kanazawa, Japan, November 2000. IEEE Computer Society.

[12] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. *Proceedings of the IEEE International Conference on Neural Networks*, pages 586–591, 1993.

[13] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, New York, 1962.

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Parallel Distributed Procesing*, volume 1, chapter 8. MIT Press, Cambridge, MA, 1986.

[15] N. F. Schneidewind. Body of knowledge for software quality measurement. *IEEE Computer*, 35(2):77–83, February 2002.

[16] E. Troubitsyna. Integrating safety analysis into formal specification of dependable systems. In *Proceedings of the International Parallel and Distributed Processing Symposium*, page 8pp. IEEE Computer Society, April 2003.

[17] Z. Xu and T. M. Khoshgoftaar. Software quality prediction for high assurance network telecommunications systems. *The Computer Journal*, 44(6):557–568, December 2001. British Computer Society.

[18] Z. Xu and T. M. Khoshgoftaar. Application of fuzzy rule extraction to minimize the costs of misclassification in software quality modeling. In J. Lee, editor, *Software Engineering With Fuzzy Theory*. Physica Verlag, 2002.

[19] M. C. Yovitz, G. T. Jacobi, and G. Goldstein. *Self Organizing Systems*. Spartan Books, Washington, DC, 1962.

Evaluating the Accuracy of Call Graphs Extracted with the Eclipse CDT

Nicholas A. Kraft and Kevin S. Webb
Department of Computer Science
The University of Alabama
Tuscaloosa, AL 35487, USA
{nkraft,kwebb}@cs.ua.edu

Abstract

The Eclipse open source development platform has garnered significant attention in recent years, largely due to its extensible plug-in architecture. Many plug-ins that analyze program source code have been developed in academia and industry; the majority of these plug-ins have used the Java Development Tools (JDT) to analyze Java source code. However, Eclipse also provides the C/C++ Development Tooling (CDT), which has now reached version 4.0. If the CDT is to become as successful a basis for implementations of static analyses as the JDT, then the challenges of using the CDT to build such static analyses must be investigated. In this paper we present an evaluation of the accuracy of call graphs extracted with the CDT. We present a CDT-based subject system and a gcc-based oracle system, and we evaluate the subject system using the oracle system. Our evaluation gives special attention to features of C programs that often cause difficulty for lightweight parsers, such as the one provided by the CDT. These features include function pointers, macros, and conditional compilation. Our results identify areas where the CDT still needs improvement, but also demonstrate the feasibility of using the CDT as a basis for more advanced static analysis tools.

1. Introduction

Most modern integrated development environments (IDEs) use a *plug-in architecture*, that is, an architecture in which tool builders can augment the base functionality of the IDE by defining extension modules (or plug-ins). Eclipse [4], Microsoft Visual Studio [16], and NetBeans [19] are all examples of IDEs that use this architecture. Many researchers have exploited these IDEs, especially Eclipse, to implement new techniques, such as static analyses of program source code. Because these IDEs provide commonly required infrastructure components, including parsers and static representations of source code, researchers can focus their efforts on implementing their techniques. In addition, these IDEs allow researchers to more easily distribute their work to practitioners as well as to other academicians.

Eclipse is an open source development platform that has garnered significant attention in recent years — largely due to its extensible plug-in architecture. Dozens of publicly-available and commercial plug-ins are available for Eclipse, including nearly 40 for analyzing program source code.Perhaps the most well known feature of Eclipse is its Java IDE, the Java Development Tools (JDT); most, nearly all, Eclipse plug-ins that perform static program analyses are based on the JDT. However, Eclipse also provides a C/C++ IDE, the C/C++ Development Tooling (CDT) [5], which has recently become more robust and has now reached version 4.0.

The CDT provides two parsers (for C and C++, respectively), each of which is hand-written with recursive descent technology, performs neither type-checking nor semantic analysis, and constructs a static internal representation from program source code. The internal representation for each language models common language extensions, such as GNU and Microsoft extensions, through the use of special, dialect-specific nodes. The CDT also provides an indexer that attempts to resolve all bindings, although binding resolution is not based on language semantics and is not as accurate as type-checking or semantic analysis.

If the CDT is to become as successful a launching point for implementations of static analyses as its breatheren the JDT, then the challenges of building such static analyses using the CDT must be investigated. In this paper we leverage the CDT to extract call graphs from the source code of C programs and evaluate the accuracy of the extracted call graphs by comparing them to call graphs extracted with an oracle system. We selected the call graph for this accuracy evaluation because it is a fundamental static program analysis that is needed to perform more advanced analyses such as interprocedural data flow analysis. This evaluation will be useful to other researchers who wish to understand the

current capabilities and shortcomings of the CDT with regard to the implementation of static program analyses.

The rest of the paper is organized as follows. In Section 2 we provide background information about the program representations and the tools that we use to perform our evaluation. In Section 3 we provide an overview of the CDT–based system that we evaluate and the gcc-based system that we use as our oracle. In Section 4 we describe our accuracy evaluation and report our results. Finally, we discuss related work in Section 5 and conclude in Section 6.

2. Background

In this section we briefly describe the program representations and the tools that we use to perform our evaluation. The three program representations that we describe are *static program representations*, that is, they can be constructed from the program source code without using dynamic (run-time) information. The tools that we describe, along with the CDT, serve as the bases for the tools which we developed to perform our evaluation.

2.1. Static Program Representations

An *abstract syntax tree* (AST) is a pruned parse tree from which nonterminals, keywords, punctuation, and other nodes and edges that do not affect the semantics of the program have been omitted. The AST is constructed by most parsers in lieu of an unabridged parse tree and is the most pervasive static program representation. An *abstract semantic graph* (ASG) is an adorned AST to which nodes and edges that represent semantic information about the program, such as edges from variables to their declarations and edges from type uses to their definitions, have been added. The ASG is often the output of a compiler front end and contains a wealth of information — one can construct many other static program representations using only the information found in an ASG [11].

A *call graph* is a directed graph in which the nodes represent the functions in a program and the edges represent the potential calls to those functions. For example, given two functions f_0 and f_1, an edge (f_0, f_1) appears in the set of edges if there is a potential call to f_1 by f_0. The call graph for a program is a directed acyclic graph (DAG) if the program does not use recursion. A call graph extracted from program source code is called a *static call graph*, while a call graph extracted from a running program is called a *dynamic call graph*.

In strictly first-order procedural languages, call graph extraction is straightforward; at each call site the target of the call is directly evident from the source code. However, in C (and C++), function pointers complicate call graph extraction. Semantic information, such as type definitions and bindings, are required to determine the target of a call made through a function pointer. Furthermore, the lax syntactic rules of C can make calls made through function pointers difficult to distinguish from normal calls if certain conventions are not followed.

2.2. GENERIC

The C/C++ compiler from the GNU Compiler Collection, gcc, uses an ASG to facilitate recognition, analysis, and optimization of a program. Since version 3.0, gcc has provided, via a command line flag, a facility for writing the ASG for a translation unit to a text file. The schema for the ASG representation stored in these text files, known as TU files, is called GENERIC [15]. Several researchers have used instances of GENERIC to perform program analysis, comprehension, and visualization tasks [1, 9, 10, 14].

The g^{4}re tool chain [12] includes a library, generic, that provides parsing, storage, traversal, and serialization facilities for working with GENERIC ASG instances. The input to generic is a TU file, and the output is a gzipped XML encoding of the input file or an in-memory representation of the ASG. The TU file parser is implemented with a flex scanner. A simple node list data structure stores the in-memory representation, and several parameterized methods traverse the leftmost child right sibling (LCRS) tree that underlies the ASG.

3. Systems

In this section we describe the two systems that we created for our evaluation: the *subject system* and the *oracle system*. In Section 3.1, we describe the subject system, which uses the AST provided by the CDT to extract call graphs. In Section 3.2, we describe the oracle system, which uses the ASG provided by gcc and the generic library to extract call graphs. Both systems extract call graphs for C programs.

3.1. Subject System

Figure 1 illustrates our subject system, which is an Eclipse plug-in that takes as input a C program consisting of one or more C files and produces as output an XML file containing a call graph. The C files, shown in the upper left of the figure, are read by the CDT, which is shown in the lower left of the figure. For each C file, the CDT produces one AST; the CDT provides a Visitor [7] base class from which other plug-ins can inherit to traverse the in-memory representation of each provided AST instance.

The cgce plug-in, shown in the lower right of Figure 1, extends the CDT and other essential Eclipse components (some of which are elided from the figure for clarity). The

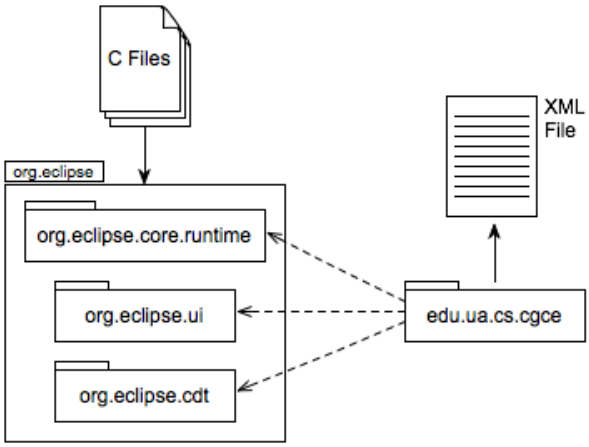


Figure 1. Overview of Subject System (cgce).
Dashed lines represent "use" dependencies. Solid lines represent data flow. Tabbed files are provided by the user. Lined files are generated by the system.

cgce plug-in extends the Visitor base class provided by the CDT to perform its AST traversals; during the traversals, cgce collects and stores information about functions and function calls. The CDT provides AST instances, not ASG instances, so cgce must manage a symbol table to resolve names; we built a partial symbol table to store information about scopes, functions, and function pointers. The symbol table that we manage allows cgce to detect calls to function pointers that the CDT call hierarchy cannot; in addition, the performance impact introduced by the symbol table is negligible.

After processing the information gathered during the visit to create a call graph, the cgce plug-in writes the call graph to an XML file, which is shown in the upper right of Figure 1. The XML file stores the call graph hierarchically by file and function; optionally, cgce can filter the call graph by either of these criteria before the XML file is written. In addition, cgce can filter the call graph based on the location of the callee function.

The cgce plug-in is full-featured and provides much more functionality than is described in the previous paragraphs. Using cgce, a user can, within Eclipse, explore the call graph in a hierarchical view or double-click the hierarchical view to jump to the definition of a function or to a function call. In addition, cgce can produce a dot [8] file that can be used to graphically view the call graph outside of Eclipse.

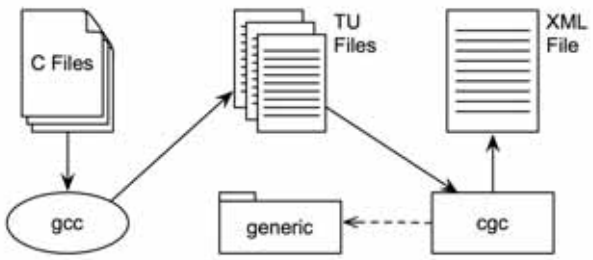


Figure 2. Overview of Oracle System (cgc).
Dashed lines represent "use" dependencies. Solid lines represent data flow. Tabbed files are provided by the user. Lined files are generated by the system.

3.2. Oracle System

Figure 2 illustrates our oracle system, which takes as input a C program consisting of one or more C files and produces as output an XML file containing a call graph. The C files, shown in the upper left of the figure, are read by gcc, which is shown in the lower left of the figure. For each C file, gcc produces one TU file; the resulting set of TU files, shown above center in the figure, is the input to the cgc program.

The cgc program, shown in the lower right of Figure 2, uses the generic library, which is shown below center in the figure, to read TU files. The Visitor pattern [7] is used by cgc to traverse the in-memory representation of each ASG instance that it obtains from the generic library. During the traversals, cgc collects and stores information about functions and function calls. After processing this information to create a call graph, cgc writes the call graph to an XML file, which is shown in the upper right of Figure 2. The XML file format produced by cgc is the same as that produced by the subject system; the filtering capabilities of cgc are also the same as those provided by the subject system.

4. Accuracy Evaluation

In this section we describe our accuracy evaluation of call graphs extracted with the CDT. In Section 4.1 we describe our technique, including the scope of our accuracy evaluation and the rationale for our oracle system. In Section 4.2 we describe our experimental setup. Finally, in Section 4.3 we list the results of our experiments.

4.1. Technique

Researchers have given much attention to extracting accurate and precise call graphs for the C language [3, 17].

Were it not for function pointers, call graph extraction for the C language would be straightforward; thus, this attention has focused on function pointers. However, before techniques for extracting call graphs in the presence of function pointers can be applied, accurate information about functions, function pointers, and function calls (including those through function pointers) must be obtained. In this evaluation, we focus on determining whether this (accurate) information can be obtained using the CDT.

In this evaluation, our oracle system, cgc, is gcc-based; in particular, cgc extracts call graphs for C programs using the ASG provided by gcc. Our rationale for designating cgc as the oracle system is based on an argument similar to the one used by Murphy, et al. [18]. The argument is as follows: during compilation, gcc builds an ASG and uses that ASG to generate executable code; if the generated executable code is correct then the ASG from which it was generated must be correct. Given that gcc is an industrial-strength compiler and that it is used to compile a myriad of widely-used programs, we can assume, with some confidence, that the executable code generated by gcc is correct, and, therefore, that the ASG used by gcc to generate that executable code is also correct. Because cgc simply gathers information from the ASG provided by gcc, we can reasonably designate it as our oracle system.

In addition to using our oracle system to evaluate the accuracy of our subject system, we used the call hierarchy view included with the CDT to validate our subject system. There are limitations to the functionality of the call hierarchy view. For example, it does not build a call graph for the entire program, but rather for one function at a time. To ensure that we were correctly extracting all available information from the CDT, we used the call hierarchy view as a reference during development of our subject system. However, please note that, as described in Section 3.1, our subject system manages a partial symbol table. Thus, our subject system is able to extract more information about calls to function pointers than the call hierarchy view.

4.2. Experimental Setup

We performed all experiments on a Dell™ OptiPlex™ 755 workstation on which we installed the Slackware 12.0 operating system. We created all TU files with gcc version 3.3.6. We used Eclipse version 3.3 and Java version 1.6.0_01 to develop and to execute our Eclipse plug-in. We wrote an XSLT stylesheet to extract results from the XML output files; we applied the stylesheet with Saxon-B version 8.9. Finally, we formatted and compared the resulting data using a series of simple bash scripts.

We use the Apache HTTP Server [2], version 2.2.6, as the test case for our evaluation. We chose Apache because it is mature, open-source, widely-used, and makes extensive use of function pointers, which are a key element of our evaluation. In addition Apache makes extensive use of macros and conditional compilation, both of which are often difficult for lightweight parsers, such as those included in IDEs, to handle properly.

	System	Subject	Oracle
User Functions	Defined	2,497	2,497
	Undefined	21	21
	Subtotal	2,518	2,518
Library Functions	Subtotal	671	666
Functions	Total	3,189	3,184

Table 1. Function Declaration Information. *The number of user functions and the number of library functions reported by each system. A function is a user function if it is declared in one of the source files for the test case and a library function if not.*

The Apache source tree contains multiple support libraries, but we selected only files from the `modules`, `server`, and `os` directories. Moreover, we only selected Unix-specific files from the `modules` and `os` directories and only selected files specific to the `prefork` version of the `mpm` module. Finally, we did not select any test modules from the `modules` directory. When generating the TU files for the oracle system, we configured Apache to include all available features.The test case resulting from our selections and configuration consists of 212 files, including 61 header (`.h`) files and 151 source (`.c`) files. The total size for the test case is approximately 89K lines of non-commented, non-preprocessed lines of source code (NCLOC).

As discussed in Section 3.2, the XML file format produced by the oracle system is the same as that produced by the subject system; furthermore, the filtering capabilities of both systems are also the same. Before performing our experiments we configured each system to remove from its extracted call graph all functions declared outside of the 212 files that comprise our test case, including all function calls found within those functions. Note that we did not remove calls to functions declared outside of the 212 files in the test case as long as the function in which the calls were made was not removed.

4.3. Results

Table 1 lists data about function declarations reported by the subject system and the oracle system. A function declaration is categorized as a user function if it is declared in one of the 212 source files for our test case; otherwise, it is categorized as a library function. Furthermore, a user function is categorized as defined if a corresponding function body was found or as undefined if no corresponding func-

	System	Subject	Oracle
Calls to Functions	User	6,884	6,884
	Library	8,842	8,809
	Subtotal	15,726	15,693
Calls to Function Pointers	Subtotal	515	525
Calls	Total	16,241	16,218

Table 2. Function Call Information. *The number of calls to functions and the number of calls to function pointers reported by each system. A call to a function is categorized as user or library depending on whether it was a call to a user function or a library function.*

tion body was found.

The data in Table 1 indicates that the CDT-based subject system is in near total agreement with the gcc-based oracle system with respect to the nodes in the call graph. The only discrepancy is in the number of reported library functions; this discrepancy is a result of the way that the subject system detects library functions. The subject system reports information about a library function only when that function is the target of a reported function call. Thus, the discrepancy in the data from Table 1 is an artifact of a misreported function call(s); a more detailed explanation follows.

Table 2 lists data about function calls reported by the subject system and the oracle system. A function call is categorized as a call to a function if the target of the call is a user or library function; otherwise it is categorized as a call to a function pointer. Furthermore, a call to a function is categorized as user if the target of the call is a user function or as library if the target is a library function.

The data in Table 2 indicates that there are function calls which the subject system does not properly recognize. In particular, the subject system reports 33 extra calls to library functions and 10 too few calls to function pointers. Upon further investigation, we discovered that some of these differences were not the fault of the subject system, but rather of the oracle system. These differences all resulted from inlining; in total, 24 calls to `strlen` with a literal string parameter were inlined by gcc. However, we also discovered flaws in the CDT centered around function pointers.

Next we discuss a C construct that is problematic for the CDT and, hence, our subject system. On line 990 of the file `server/vhost.c`, a parameter `func_cb`, to the function `ap_vhost_iterate_given_conn` has type `ap_vhost_iterate_conn_cb`, which is an alias type (`typedef`) for a function pointer. A call to `func_cb` on line 1024 is not recognized by the CDT as a function call; however, our subject system does correctly determine that there is a function call on line 1024. Unfortunately, due to the limited information in the partial symbol table, our subject system does not correctly report the call as a call to a function pointer, but rather reports the call as a call to a library function. Calls to function pointers that are passed as parameters account for six (6) of the extra calls to library functions listed in Table 2. In addition, such calls account for the five (5) extra library functions listed in Table 1.

We have accounted for 30 of the 33 extra calls and 6 of the 10 too few calls to function pointers reported by our subject system. The remaining three extra calls are caused by problems the CDT has dealing with a combination of function pointers and macros, namely with a macro expansion that defines a function pointer variable and a later call through that variable. Curiously, this problem only appeared three times, and was properly handled in other cases. Nevertheless, this combination of a macro and function pointer accounts for the three remaining extra calls to library functions and three of the four remaining too few calls to function pointers reported by our subject system. As in the previous paragraph, while our subject system misidentified these calls as calls to library functions, the CDT did not recognize the function call at all.

Finally, on line 106 of the file `server/util_pcre.c`, there is a call of the form `(pcre_free)(preg->re_pcre)`. The CDT does not recognize the function call, and the partial symbol table managed by our subject system lacks the information necessary to identify the call. This is due to the function pointer `pcre_free` being defined in a library header file, which our subject system ignores. We have now accounted for all of the differences listed in Table 2.

5. Related Work

In this section we briefly review literature that relates to our work. There are two primary categories of research about call graphs: (1) extraction and (2) evaluation. We briefly review this research, and we then review tools that use the CDT to extract call graphs.

Milanova, et al. [17] apply the FA pointer analysis by Zhang et al. [21] to the extraction of call graphs for C programs with function pointers. They compare a call graph extracted using this analysis to the most precise call graph that can be extracted with existing pointer analyses. They conclude that the FA analysis can provide precision comparable to that of more expensive pointer analyses. Atkinson [3] describes a technique to improve the precision of extracted call graphs for C program with function pointers. The technique is parameterized by a pointer analysis and is safe (it does not degrade the accuracy of the extracted call graph). The supplied pointer analysis is augmented by function type information, and potential calls through function pointers are filtered based on the results of the (improved) pointer analysis. The focus of both of these techniques is on improving the precision of extracted call graphs, not on extracting the accurate information needed to implement these

techniques.

Murphy, et al. [18] describe the most comprehensive study of static call graph extractors to date. They empirically compare call graphs extracted from three software systems written in C by nine call graph extractors and present both quantitative and qualitative findings. In addition, they define a design space for static call graph extractors. Their argument for their oracle system is similar to our argument for our oracle system; however, their oracle system is not based directly on a compiler. Instead, their oracle system is based on a source-to-source translation tool that is based upon an obsolete version of gcc.

Lhoták [13] contributes a call graph difference search tool. The tool ranks edges in the call graph by their likelihood of causing significant differences among call graphs. In addition, the author describes a complimentary viewer that highlights certain differences among call graphs. Furthermore, the author presents the ranking algorithm implemented by the first tool and describes the results of a case study. The focus of the case study is determining imprecision in a static call graph by comparing it to a dynamic call graph for the same program; the subject language for the case study is Java.

We are aware of two other tools based on the CDT that extract call graphs for C programs. The first of these tools is the CDT itself. The most recent version of the CDT, version 4.0, includes the call hierarchy view, which shows the user the callers of a function. However, the call hierarchy view operates on only one function at a time – it does not build a call graph for the entire program. The second tool is the Eclipse Parallel Tools Platform (PTP) [6]. Recoskie and Tibbitts [20] reported on the static analysis capabilities of PTP, including a call graph extractor. However, they did not report on the accuracy of the call graphs that they extract.

6. Conclusions and Future Work

In this paper we presented an accuracy evaluation of call graphs extracted with the Eclipse CDT. We built a subject system and an oracle system, and we evaluated the subject system using the oracle system. In our evaluation, we used the Apache HTTP Server as our test case, because it is open-source, is mature, is widely-used, and makes extensive use of function pointers, macros, and conditional compilation. Our evaluation can be viewed as a (partial) accuracy evaluation of the AST produced by the CDT, and therefore, as a (partial) accuracy evaluation of the C parser from the CDT. The results of our evaluation show that the parser is extremely robust, even in the presence of heavy use of function pointers and the preprocessor.

We plan to use the CDT to build a more comprehensive source code analysis tool for the C language. We envision a general purpose tool that will complement the existing features of the CDT, as well as the Eclipse Parallel Tools Platform (PTP). Our goal is to facilitate comprehension and maintenance tasks of large software systems written in C, including the Linux kernel and the GNU Compiler Collection.

References

[1] G. Antoniol, M. D. Penta, G. Masone, and U. Villano. Compiler hacking for source code analysis. *Software Quality Journal*, 12(4):383–406, December 2004.

[2] Apache HTTP Server Project. http://httpd.apache.org/.

[3] D. C. Atkinson. Accurate call graph extraction of programs with function pointers using type signatures. In *APSEC*, pages 326–335, 2004.

[4] Eclipse – An Open Development Platform. http://www.eclipse.org/.

[5] Eclipse C/C++ Development Tooling. http://www.eclipse.org/cdt/.

[6] Eclipse Parallel Tools Platform. http://www.eclipse.org/ptp/.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[8] Graphviz – Graph Visualization Software. http://www.graphviz.org/.

[9] T. Gschwind, M. Pinzger, and H. Gall. TUAnalyzer - analyzing templates in C++ code. In *WCRE*, pages 48–57, 2004.

[10] B. N. Hoipkemier, N. A. Kraft, and B. A. Malloy. 3D visualization of class template diagrams for deployed open source applications. In *SEKE*, pages 232–235, 2006.

[11] N. A. Kraft, B. A. Malloy, and J. F. Power. An infrastructure to support interoperability in reverse engineering. *Information and Software Technology*, 49(3):292–307, March 2007.

[12] N. A. Kraft, B. A. Malloy, and J. F. Power. A tool chain for reverse engineering C++ applications. *Science of Computer Programming*, 69(1–3):3–13, December 2007.

[13] O. Lhoták. Comparing call graphs. In *PASTE*, pages 37–42, 2007.

[14] B. A. Malloy and J. F. Power. Exploiting UML dynamic object modeling for the visualization of C++ programs. In *SoftViz*, pages 105–114, 2005.

[15] J. Merrill. GENERIC and GIMPLE: A new tree representation for entire functions. In *Proceedings of the GCC Developers Summit*, pages 171–180, 2003.

[16] Microsoft Visual Studio. http://msdn.microsoft.com/vstudio/.

[17] A. Milanova, A. Rountev, and B. G. Ryder. Precise call graphs for C programs with function pointers. *Automated Software Engineering*, 11(1):7–26, January 2004.

[18] G. C. Murphy, D. Notkin, W. G. Griswold, and E. S. Lan. An empirical study of static call graph extractors. *TOSEM*, 7(2):158–191, April 1998.

[19] NetBeans IDE. http://www.netbeans.org/.

[20] C. Recoskie and B. Tibbitts. C/C++ source code introspection using the cdt, March 2007. Presentation at EclipseCon 2007.

[21] S. Zhang, B. G. Ryder, and W. Landi. Program decomposition for pointer aliasing: A step towards practical analyses. In *FSE*, pages 81–92, 1996.

A comparison of time tracking tools for software developers

Jouni Lappalainen, Lasse Harjumaa, Jukka Sirviö, Tytti Pokka,
Heidi Moisanen and Hanna Leskinen
University of Oulu, Department of Information Processing Science
P.O. Box 3000, FIN-90014 OULUN YLIOPISTO
jouni.lappapainen@oulu.fi

Abstract

This paper introduces a case study that was carried out in a small Finnish software development organization. The aim of the study was to evaluate possibilities for establishing tool support for tracking software developers' working time. The organization wants to improve their project management and effort estimation processes and they need a lightweight, tool-based approach for collecting and reporting the working time data. We propose a categorization for different types of time tracking tools and perform an evaluation of tools from different categories. Requirements for the tools and evaluation criteria are derived both from literature and the case company. DESMET method is utilized to rank the tools.

1. Introduction

Software measurement aims at providing quantitative information about development processes in order to estimate effort, time and costs that are required to produce a software product. The measurement process itself includes a wide variety of activities and specific metrics. Resource estimation is one of the first key elements to implement when establishing a predictable software development process.

Estimates are typically based on historical data, and recording, storing and analyzing process data is a challenging task. Ensuring the accuracy of the data is even more difficult, because different people can report the same data differently. Software development is human-centric, creative work, and measuring such phenomenon always requires some assumptions to be made. However, measurement is a necessity if an organization wants to systematically manage and improve their development processes.

Software tools can make the resource estimation task easier by automating calculations and providing means for collecting and storing the data [8]. This paper gives an overview to time tracking tools that are available for software developers. Furthermore, we introduce a classification of time tracking tools and present an evaluation of selected tools with the DESMET method. The tool evaluation is a part of a case study investigating deployment of software measurement tools within a small software development organization.

The rest of the paper is organized as follows. Chapter two introduces the research that have been done in the area of time tracking tools. Chapter three describes the research setting of this study.

2. Related work

Version control systems and bug tracking databases have been identified useful tools for getting information of software development projects in order to make effort estimations. [1, 13] Reliable effort estimation is crucial for estimating cost and schedule of a project, and helps in risk assessment. For cost estimation, several expert-based techniques [11] and algorithmic models, such as COCOMO [2] have been introduced. Effort estimation models can also have indirect benefits in predicting the number of defects in software and helps in making decision when to stop testing, for example [14]. Algorithmic estimation models are usually addressed to large projects. More lightweight methods exist for making estimates in small or individual-based projects. Such methods are the Personal Software Process (PSP) [5] or LEAP toolkit [6], for example.

Several studies suggest that establishing a software measurement successfully requires an adequate tool support. Tools for automatic data collection and use of persistent measures database are often listed among

the essential success factors of a measurement framework. Examples of the articles and studies highlighting the importance of correct tools for measurement include [4, 17], for example.

Especially the PSP approach has given inspiration to many time tracking tools, including Process Dashboard (http:// processdash.sourceforge.net/), PROM [19], Hackystat [9] and Jasmine [18]. In PSP, measurement and analysis of historical development data is in key role in making estimates of effort and product quality.

There is a wide variety of time tracking and effort estimation tools. Even a spreadsheet application can provide the basic set of templates and reports. In the other end are sensor-based tools that can collect data automatically and provide an extensive selection of analysis reports based on the data. It may also be useful to integrate process guidance and experience base into the measurement tool [18].

Johnson et al. [6] introduce three generations of tool support for gathering metrics on individual software developers' work. The first generation is manual data collection, developers logging their effort, size and defect information in printed templates and forms. The second generation includes more automated tools such as Process Dashboard, where users can record data through a relatively convenient user interface and the tool provides different types of analyses on the data. The third generation means fully automated measurement, in which developers do not have to worry about data logging or analysis.

Tools can also have some disadvantages, such as context switching. The need to continuously change between product development and process recording software may cause inaccuracies in the data and decrease the motivation to use the tools. Fully automated data collection tools eliminate the need for context switching. However, fully automated tools cannot necessarily recognize breaks in developer's work correctly, and it may be needed to correct automatically recorded data later on. Concerns on privacy is another challenge with automation. [8]

3. Research setting

The case organization is a small division of a large multinational company developing embedded systems. The division that participated in the study is a software development unit of about 20 software designers.

A study project for investigating time tracking systems was established because the organization wants to forecast the durations of their future projects. The products of the company are getting more software intensive and more attention has to be paid to the software development processes than before.

The purpose of the study was to investigate commercial and freeware solutions available for tracking the work time of software developers. Currently the organization utilizes simple time card system for that purpose. It is connected to the access control so that on arrival an employee clocks in for a specific project number and the system marks the working hours for that project. This system, and the information it provides, is not adequate for the company's needs in future. The current system stores only information about the projects the employees are currently working on and how many hours they have spent on each project. Even though the current system keeps record of the working time on the project-basis, the information is not detailed enough to really help in planning future projects and estimating project schedules and costs. The current system does not support reporting on what specific tasks of the project designers are doing. Such information would be necessary in order to monitor the development effort and identifying possible problems in the development process. For example, distinguishing design, implementation and testing efforts from each other is essential for evaluating the quality of both product and process.

The case organization wishes to establish a historical development database with the new time tracking system. Based on the historical data, schedule and effort estimates could be made more accurately than at the present. Furthermore, the estimation can be targeted more precisely regarding the phases of the development and individual tasks within a project.

For the purposes of this study the reference framework used in the tool comparison is DESMET [12]. It is a two-part methodology for comparatory evaluation of tools and methods in a given context for a specific purpose. The first part of the DESMET method includes the selection of an appropriate evaluation method, and the second part the actual methods and their descriptions. From the quantitative, qualitative and hybrid evaluation methods available we chose the qualitative feature analysis, which was most feasible given the research setting.

Although the actual tool tests weren't designed and conducted as a part of a process utilizing the DESMET method in the first place, it is used here to structure the empirical findings. On the other hand, the feature analysis selected as the evaluation method is usually conducted based on the literature and documentation available for the evaluation object. Therefore this

study, while using the test reports from the case company tool evaluation tests, can be seen as fulfilling the signs of a DESMET feature analysis.

It should be noted that the DESMET is designed for the industry, to aid in selecting the best suited tool or method for that context from the given set of candidates. In this case the context is an industrial case which is also a source for a part of the tool requirements, so the use of the DESMET method is justifiable also in this academic study.

4. Features used in evaluation

From the nine basic processes included in a feature analysis, the tool selection and its results are described in more detail in the next section. This chapter discusses the selection of required features for the tools to be evaluated, their prioritization and their scoring system. The level of rigour required cannot be affected in this study, since the only available source of information for the evaluator is the written test documentation. Responsibility of carrying out the evaluation is rather trivial in this case, and the analysis of the evaluation results is presented in the results and discussion section.

As feature analysis supports feature groups, hierarchies and refinements, this study uses it to classify the requirements on the main level to two different groups: those identified by the case company and those identified by the test team with the aid of some scientific literature. [e.g. 3, 7, 8, 9, 16] Summary of the requirements – which have been considered as features that are studied from each tool in this study –

Table 1. Case-company originated features for a tool to be evaluated in feature analysis.

Classification	Description	Priority
Data mgmt.	Access to database is controlled	1
Data mgmt.	Database stores relevant modifiable raw variables	1
Data mgmt.	All data stored in a web/intranet-server database	1
Functional	Free choosing of project and phase worked on	1
Functional	Worktime input must be possible also “after-the-fact”	1
Functional	At least worktime calculations must be automated	1
Functional	Tool must support customisable projects and phases in them	1
Functional	Tool must support recording and reporting times/phase, times/project, times/user	1
Functional	Time granularity must be represented accurately (hours and minutes)	1
Functional	Support for multiple simultanious users	1
Functional	Privacy of personal data among developers	2
Functional	Both estimated and actual times are supported and they can be compared	2
Interface	Time tracking must be done by pushing start/pause/stop -button(s)	1
Interface	Support for existing operating systems	1
Interface	Support for multiple common file formats (XML, HTML, txt)	1
Interface	Compatibility with existing systems	2
Reporting	Reports are accurate, stable and reliable	1
Reporting	Data collected: project, phase, people, start time/phase, end time/phase, total times	1
Reporting	Reports based on historical data	1

Table 2. Literature-based features for a tool to be evaluated in feature analysis.

Classification	Description	Priority
Functional	Data is modifiable and importable/exportable	1
Functional	System informs the user about status and events	2
Functional	System should support mistake-free data collecting	2
Interface	System should be customizable for expected types of changes	2
Usability	System should not allow the user to forget the use of the system	1
Usability	Tool usage must be as unobtrusive as possible (few minutes / day)	1
Usability	High overall usability, especially on the features used most often.	1
Usability	minimum need of investments and involvment from management	2
Nielsen	System should speak the user's language	1
Nielsen	Consistent use of terms and concepts.	1
Nielsen	Minimize the amount of things the user needs to remember	1
Nielsen	Flexibility for tailoring, efficiency for both beginners and experts	1
Nielsen	preventive error -approach	1
Nielsen	provide help and documentation	1
Nielsen	Provide easy exits	2
Nielsen	Informative error messages	2
Nielsen	Minimalistic design	2
Nielsen	give feedback to user	2

are presented in Table 1 and Table 2. Each feature is also attached with a scoring scheme, against which each evaluated tool is ranked. For this study, a simple yes/no scoring scale is adopted, indicating just the presence of a feature in the tool without analysing the level of the implementation of the feature. Priorities (indicated with 1 and 2 in Tables 1 and 2. Priority 1 is of higher priority) for each feature were devised along the requirements by the case company.

Both main categories of features are further divided into feature classes. These classes indicate if the feature in question is mainly related to the domain area –functionality of the system, how the data is managed by the system, what kind of interfaces the system has to other systems as well as the user, how the system handles reporting of its data to different users, and what kind of usability the system has. Generic usability heuristics are incorporated into these features in the form of Nielsen's heuristics [15, 16].

5. Tool evaluation

For any evaluation framework to produce usable results, an object of analysis is required. In this case the DESMET method being comparative, a selection of different tools is used. Based on initial set of requirements (simple, multi-user tool for Windows platform that has a free trial version available), some tools were searched, found and selected to be evaluated. By all means all possible project management and time tracking systems were not tested, from initial searches ten tools were selected for further testing.

For this study, a representative sample of those ten tools was attempted to find. This was done by categorizing the tools according to their method of gathering data, as per classification devised by the testing [8, 10]. The categories for the tools were based on the way the systems gather their data, the so-called level of time tracking: manual, system-based, recording-based or fully automatic tracking.

Manual tracking consists of methods that involve the user writing times manually into some form of log, like a timesheet, and then delivering those timesheets to her supervisors. It has the advantage of being a very flexible method, but also error-prone and time-consuming way of tracking data.

System-based tracking is partly automated way of data gathering, where each user submits her timesheet to a system that allows supervisors to generate reports from user data. It has the benefit of being more real-time than completely manual method and fairly cheap as well as easy way, but suffers from potential compatibility problems with existing systems and its reliability is in the end down to the user and how accurately she fills her timesheets.

Recording-based tracking can also be described as start-pause-stop -tracking, due to the fact that systems that incorporate this method usually do so by providing user with a user interface to start, pause and stop the tracking with a single press of a button. From the state of the tracker timesheets are automatically filled, and reports generated as in system-based tracking. Advantage of this method is that it provides somewhat more reliable user data than system-based trackers, since the filling of the timesheet is done more simply and promptly when relevant – there's no need for user to remember exact times when the system does it automatically. As with any method, this also has some drawbacks: Since it may distract the user from other, more thought-intensive tasks at the time, the motivation to use the system may not be high, thus causing inaccuracies in collected data. Also the monitoring of the time spent working away from the computer is difficult, and usually must be done in a way very similar to the system-based tracking method.

Completely automated tracking systems usually function by attaching sensors or similar hooks to different software tools that are used by the developers as part of their development work. From the usage statistic acquired via those sensors time and possibly also other metrics can be calculated. On the obvious positive side this way of collecting data is completely unobtrusive and therefore suffers no inaccuracies due to use of the tracking tool itself, but is more inaccurate in its way of gathering off-computer working time. In addition, the problem of resolving the quality of actual work done (if any) while the sensors register activity in tool usage remains problematic.

From the above categories, the following tools were selected for this evaluation as being the most representative of its class: For system-based trackers and as sort of a "de-facto baseline" MS Project is evaluated. For SPS-systems the Process/Team Dashboard is used for evaluation, and Dovico Timesheet is included as a system combining the system- and SPS-approaches. Automatic systems weren't tested with the case company, and thus are evaluated here based solely on academic and possibly other literature, are represented by the University of Hawaii's Hackystat. For completely manual way of tracking data no tool is selected – since it doesn't necessarily use any.

Table 3. Total features passed for each tool grouped by feature category.

	MS Project	Dashboard	Dovico	Hackystat	Max. available
Functional	8	12	12	9	12
Nielsen	6	8	9	8	10
Interface	2	5	5	4	5
Usability	1	3	3	4	4
Database	0	0	3	2	3
Reporting	2	3	3	2	3

6. Results and discussion

The last phases of the feature analysis consist of analyzing the results and presenting them to interested parties. In this case the presentation is done via this paper, and the analysis we will provide here. All of the five tools in this evaluation were assigned a true/false value for each of the features identified to be relevant. There were no predefined acceptance threshold of values, but it was understood that the tool that had the most desired features supported was the one that thecase organization would consider to be selected. The results are portrayed in table 3. Each row in the table represents one class of features that were analyzed. On the columns for those rows are the numbers of features that were present in each tool, columns therefore representing the tools in the evaluation. One of the columns is labeled as "Max. available", which represents the maximum number of features that were analyzed in each category. Compared to these maximum values we can see that none of the tools fulfils the requirements completely. Biggest deviations from the maximum values are in the database and usability feature categories. These are the topics that the tool developers should pay attention to in order to make their products more appealing to the industrial users. On the other hand all tools scored well in the functional and reporting feature categories.

When comparing the individual tools, it is evident that MS Project that is probably the most widely-used project management tool does not compare well to the maximum values. Full automation as provided by the Hackystat tool has also some shortcomings. These problems stem from the fact that you cannot automate all aspects of time tracking. The main issue with the Hackystat tool concerns data collection capabilities. It is not easily possible to track time while user is doing work away from the computer. The Dovico Timesheet and Project Dashboard are quite similar, which is not too surprising since they represent the same SPS-tool category in this evaluation. Main differences between these two are in the database support they provide and in usability characteristics. Although usability is rather subjective measure, the support for relational databases is surely something that industrial users desire.

It seems that a well-implemented SPS-type of tool is better than current fully automated systems for time tracking, but if full automation could be implemented with more comprehensive time tracking support it would be the best solution. This, as well as better SPS-tool implementations are something that should be investigated in further studies. In this study the case organization decided to evaluate and pilot the Dovico Timesheet tool more closely. The choice was made between the Dashboard and Dovico Timesheet, but the usability of the latter was perceived to be better.

This kind of evaluation benefits the case organization, because it focuses attention to the process of effort estimation and time management, and thus improves both of them via the "Hawthorne-effect". A well-argumented tool selection fixes these processes as well to some extent, but isn't enough by itself.

Finally, the DESMET method works well for this kind of evaluation, but if more than one person is involved in defining the features and scoring them, it is very likely that issues of subjectivity and difficulties in communication arise. Especially the interpretation of the meanings behind feature descriptions has to be communicated carefully. If the communication difficulties can be overcome an evaluation using the DESMET method is very well suited for collaboration and knowledge exchange between academia.

7. Conclusion

This study draws together requirements for time tracking tools from a case company and relevant academic literature. We found that a systematic approach for evaluating the tools based on these

requirements is beneficial even though an ideal tool would not exist. Requirements definition, evaluating the tools and reflecting the entire process of effort estimation and time management helps the organization to identify potential process improvement areas.

We suggested a classification for time tracking tools, partially based on literature and partially on empirical observations during this study. Based on the experiences gained from this case, instead of manual or system-based approaches to time tracking, we suggest either a well-implemented fully automated tracking tool or a more comprehensive support via a SPS-tool to be explored when selecting a time tracking tool.

The tool support for time tracking is an actively studied area, but there remains a number of issues that need further investigation. Before a fully automated tool that covers all phases of data collection is developed, the time tracking data is as accurate as the people that track it.

References

[1] Atkins, D., Ball, T., Graves, T. & Mockus, A. Using version control data to evaluate the impact of software tools: A case study of the version editor. IEEE Transactions on Software Engineering, 28(7): 625-637, 2002.

[2] Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R. & Selby, R.. Cost models for future software lifecycles: COCOMO 2.0. Annals of Software Engineering, vol. 1: 57-94, 1995.

[3] Faulk, S., Gustafson, J., Johnson, P., Porter, A., Tichy, W. & Votta, L. Measuring HPC Productivity. International Journal of High Performance Computing, 18(4): 2004.

[4] Fenton, N. & Neil, M. Software metrics: roadmap. In proceedings of the ICSE - Future of Software Engineering Track 2000, 357-370.

[5] Humphrey, W. S. Introduction to the Personal Software Process. Addison-Wesley: Reading, MA, 1997.

[6] Johnson, P.M., Moore, C., Dane, J.A. & Brewer, R. S. Empirically guided software effort guesstimation. IEEE Software, 17(6): 51-56, 2000.

[7] Johnson, P. Project Hackystat: Accelerating adoption of empirically guided system development through non-disruptive, developer-centric, in-process data collection and analysis. Univ. of Hawaii, technical report, 2001a.

[8] Johnson, P. You can't even ask them to push a button: Toward ubiquitous, developer-centric, empirical software engineering. The NSF Workshop for New Visions for Software Design and Productivity: Research and Applications, Nashville, TN, 2001b.

[9] Johnson, P. Supporting development of highly dependable system through continuous, automated, in-process, and individualized system measurement validation. University of Hawaii, technical report, 2002.

[10] Johnson, P. Ultra-automation and ultra-autonomy for software engineering management of ultra-large-scale systems. Proceedings of the 2007 Workshop on Ultra Large Scale Systems, Minneapolis, Minnesota, 2007.

[11] Jorgensen, M. A review of studies on expert estimation of software development effort. Journal of Systems and Software 70(1-2): 37-60, 2004.

[12] Kitchenham, B. DESMET: A Method for Evaluating Software Engineering Methods and Tools. Technical Report TR96-09. Dept. of Computer Science, University of Keele, U.K., 1996.

[13] Mockus, A. & Votta, L.G. Identifying reasons for software change using historic databases. Proceedings of the International Conference on Software Maintenance, 120-130, 2000.

[14] Mockus, A., Weiss, D.M. & Zhang, P. Understanding and predicting effort in software projects. Proceedings of the International Conference on Software Engineering, 274-284, 2003.

[15] Molich, R., & Nielsen, J. Improving a human-computer dialogue, Communications of the ACM, 33(3): 338-348, 1990.

[16] Nielsen, J., and Molich, R. (1990). Heuristic evaluation of user interfaces, Proc. ACM CHI'90 Conf. (Seattle, WA, 1-5 April), 249-256.

[17] Offen, R.J. & Jeffery, R. Establishing software measurement programs, IEEE Software, 14(2): 45-53, 1997.

[18] Shin, H., Choi, H. & Baik, J. Jasmine: A PSP supporting tool. Proceedings of the International Conference on SP, 73-83, 2007.

[19] Sillitti, A., Janes, A., Succi, G. & Vernazza, T. Collecting, integrating and analyzing software metrics and personal software process data. In Proceedings of the 29th EUROMICRO Conference, 336–343, 2003.

RealSpec: An Executable Specification Language for Modeling Resources

Amir A. Khwaja
Intel Corporation
Austin, TX 78746
amir.a.khwaja@intel.com

Joseph E. Urban
Dept. of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
joseph.urban@ttu.edu

Abstract

Resource modeling is an important aspect of real-time systems. However, resource modeling and resource constraints are found to be lacking from majority of real-time specification language. RealSpec real-time specification language is proposed with language constructs for defining and manipulating both abstract data structuring and hardware system resources. The paper provides details of abstract data resources along with examples. The well known dining philosophers' problem is specified to demonstrate language resource modeling features.

1. Introduction

Resource modeling and constraints are an important aspect of real-time systems [1]. The timing behavior of a real-time system depends not only on delays due to process synchronization, but also on the availability of shared resources [2]. As with timing constraints, specifying physical resources and constraints on those resources required by a computation has a key role in determining the performance, quality, and correctness of computation in real-time systems. Even though Lee et al. pointed out the lack of resource modeling consideration in the context of process algebra formalism [2], the issue seems to be equally prevalent for most real-time specification techniques. The lack of resource modeling was also highlighted by a comparison of real-time specification techniques [3]. A real-time specification should provide a highly integrated and time-constrained resource modeling approach along with interaction protocols in a platform independent way [4, 5]. In addition, the specification should provide the primitives to control and to keep track of resource utilization.

This paper presents process and resource modeling concepts and constructs in RealSpec, a real-time specification language based on the functional dataflow language Lucid [6]. In doing so, the language semantics of Lucid are extended to include complex data types with functions, a concept necessary for defining resource objects. The selection of Lucid was made to satisfy language design goals of functional computation model, declarative nature, and freedom from implementation bias. Moreover, Lucid supports high parallelism by design with every equation within a program representing parallel executing data flow nets. The remainder of the paper is organized in the following sections. Section 2 provides a synopsis of the Lucid dataflow programming language. Section 3 briefly discusses the semantic extensions to Lucid to add user defined algebras and objects. Section 4 introduces RealSpec specification language. Section 5 presents resource modeling concepts in RealSpec. Section 6 applies process and resource concepts to the specification of the dining philosophers' problem. Section 7 summarizes the paper.

2. Lucid Dataflow Programming Language

Lucid [6] is a functional dataflow programming language based on Landin's Iswim language [7]. The statements in a Lucid program are equations defining streams and filters, not commands for updating storage locations as in the case of traditional imperative programming languages. Hence, Lucid is a definitional language. The equations in a Lucid program are assertions or axioms from which other assertions can be derived using the Lucid axioms and rules of inference [8]. Hence, Lucid provides a formal system where proofs of programs can be carried out [8]. The Lucid programmer states exactly the output of a program, but only suggests or indicates the

computations to be performed. The variables in Lucid are regarded as dynamic objects and their values are simultaneously updated at each computation step. The following is a simple Lucid program example:

```
a + b
   where
      a = 2;
      b = 5;
   end;
```

In the above program the **where** clause is an expression together with a set of definitions of variables used in the expression.

Lucid allows functions to be defined by equations where the equations within the **where** clause can use other functions or variables defined in the clause. The expression defining a function can also use the same function so that recursive definitions are also possible:

```
fac(p)
   where
      fac(n) = if n < 2 then 1
            else n * fac(n-1) fi;
   end;
```

There is no requirement that a variable or function be defined before it is used. In general, the order of the definitions in a clause is irrelevant.

The simplest conditional expression in Lucid is the primitive if-then-else. For example, the following program takes two input streams x and y and outputs a stream consisting of the maximum at each index value:

```
max(x, y) = if x <= y then y else x fi;
```

If x and y have values <5, 2, 10, 21, 7,…> and <3, 9, 9, 13, 15,…>, respectively, then the output stream would be <5, 9, 10, 21, 15,…>.

Lucid provides a binary operator **fby** (followed by) for abstract iteration over sequences. The two arguments of this binary filter are combined by taking the first component of the first argument and appending to it the stream corresponding to the second argument. For example, if x and y are two streams <x_0, x_1, x_2, …> and <y_0, y_1, y_2, …>, respectively, then x **fby** y would be the stream <x_0, y_0, y_1, y_2, …>. For example,

```
x
   where
      x = 1 fby x+1;
   end;
```

This program defines x (the output) to be the stream <1, 2, 3, 4, 5,…>, an infinitely varying sequence. The first argument of **fby** primes the pump that permits successive future values to be generated.

The binary operator **asa** (as soon as) computes by repeatedly reading in pairs of values of its two arguments until, if ever, the second argument has the value *true*. Assuming a *true* is eventually read then the value taken by the **asa** operator will be the value of the first argument corresponding to the *true* value of the second argument. For example, if x = <0, 1, 2, 3,…> and y = <false, true, false, true,…>, then

```
x asa y = <1, 1, 1,…>
```

The next section provides extensions to Lucid semantics for resource modeling.

3. User Defined Algebras and Objects in Lucid

Lucid is based on a few fixed algebras such as integers, reals, Booleans, and strings. However, in order to be able to represent resources, Lucid needs to be enhanced to include user defined algebras for representing complex data types. Based on the Iswim family of languages [7], data objects may have operator nets or filters in addition to a set of variables. A Lucid program then would be an operator net for this "data with operator net". Each instance of input and output would be some form of this data with a specific internal state based on the values of its member data types and the internal operator nets. Thus, the instance variables of a data object must themselves be full-fledged Lucid streams resulting in streams of data objects which contain a stream of data objects that are followed by potentially an n number of streams [9]. This concept is demonstrated below for a data object d with an internal instance variable x, where t′ is the time index for d stream and d_i represent object d with an updated state based on its internal instance variable and manipulation functions. For each d_i, the internal instance variable x goes through a sequence of values indexed by t″:

	t′=0	t′=1	t′=2 …
	d_0	d_1	d_2 →
t″=0	a_0 ↓	b_0 ↓	c_0 ↓
t″=1	a_1	b_1	c_1
t″=2	a_2	b_2	c_2
…			

Lucid is based on the unconventional idea that computations do not terminate. The input to a Lucid program which runs indefinitely is the infinite history of the values 'fed into' it, and the output is the infinite history of all the values produced [6]. Lucid, however, allows writing programs whose actual output is finite. To terminate a Lucid program, a special symbol **eod** (end-of-data) is output that closes the output stream and

terminates the program normally. For example, the program

```
3 fby 5 fby eod
```

will put 3 and 5 on the standard output and then terminate. The semantics of the member functions of data objects with internal streams is defined using the terminated streams. Each call to a local function will result in the data object function to be evaluated based on the local **where** clauses followed by an implicit **eod**. RealSpec uses the terminated stream semantics for data object member function calls.

4. RealSpec: An Executable Specification Language Based on Lucid

The following section introduces basic concepts of the RealSpec executable specification language.

4.1. Active and Passive Objects

Two types of objects are modeled in RealSpec: active and passive. Active objects are those which instigate an action and are responsible for handling control to other objects. Active objects can change their own states and may utilize services of other passive objects. Active objects have their own execution threads. Passive objects are those which do not have their own execution thread instead these passively wait for other objects to require their services. Passive objects are usually activated on receiving messages from other objects. RealSpec uses active objects to model system processes and passive objects to model system resources.

4.2. System Definition

A system construct is used to model a system in RealSpec. The system construct provides a context for the rest of the specification and consists of the declaration of system resources, statically defined processes, process and thread creation order, and global system level functions. A system definition in RealSpec is specified using a system construct:

```
system s {
   resources { … }
   processes { … }
   functions { … }
}
```

4.3. Process and Thread Objects

A process object is defined using a process construct. A process construct consists of the keyword **process** with process name followed by process body within curly brackets. The body of a process definition may consist of declaration for primitive data variables, other active or passive objects, and a set of process functions. The functions in a process are Lucid-like declarative assertions. For example, a process `factorial` that contains a single function calc`fac(int n)` to calculate factorial can be defined as follows:

```
process factorial() {
   calcfac(int n)
      where {
         fac(int x) = if x < 2 then 1
            else x * fac(x-1);
      }
}
```

A process has a single execution thread by default. However, a process may have as many threads as possible. Multiple threads can be defined as part of a process definition. In the example below, `x` gets the value of `x+1` if the executing thread is `th1`, indicated by the property `pid`, otherwise `x` gets the value of `x*2`:

```
process p() threads th1, th2 {
   … x = 1 fby if pid == 0 then x+1
       else x*2; …
}
```

The processes can also communicate with each other via message passing using a pair of send and receive thread functions. The message communications can either be synchronous or asynchronous. In the following example of synchronous message passing with timeouts, `p1` blocks for 50 microseconds and `p2` blocks for 75 microseconds:

```
process p1() {
   … p2.send(data) @tout 50 us; …
}

process p2() {
   … x = p1.receive() @tout 75 us; …
}
```

5. Resource Modeling in RealSpec

RealSpec supports modeling of system resources. These resources consist of abstract data structure such as semaphore, mutex, array, stack, and queue, and hardware resources such as signal, analog IO, memory, communication channel, and timer. The hardware resources are the representation of various system and hardware elements along with mechanisms for manipulation of these resources by the specification processes and functions. This paper focuses on the abstract data resources.

Users can define new resources using a resource construct. A resource construct consists of the keyword **resource** with resource name followed by resource body within curly brackets. The body of

resource consists of resource variable definitions and resource functions. The variables and functions may appear in any order.

```
resource <name>(<parameters>) {
   <resource variables>
   <resource functions>
}
```

Once defined, resources must be instantiated before these can be used. The instantiation may be at the system level, process level, or within other resources.

Semaphore and Mutex Resource: Semaphore resource is defined for process and thread synchronization. The resource is used to protect shared resources and critical regions. The semaphore resource is defined with two functions **`wait`** and **`signal`**, a private integer counter, and a private queue of threads. The parameter `value` is passed in by users as the initial counter value at declaration. The optional parameter `pol` may be used by users to define sleeping thread wakeup scheme. RealSpec supports two wakeup schemes, first-come-first-serve and priority based, defined as constants. The resource is defined using following schema:

```
semaphore(int value, int pol =
   FCFS_BASED) {
   int counter = value;
   int policy = pol;
   list pidqueue = [];
   bool wait(generic p);
   bool signal();
}
```

Consider an example where there are two rows of ballroom dancers, leaders and followers, waiting in two queues, A and B, before entering the dance floor. When a leader arrives, it checks to see if there is a follower waiting. If so, they can both proceed. Otherwise, the leader waits. Similarly, when a follower arrives, it checks for a leader and either proceeds or waits. This problem can be represented in RealSpec by two semaphores, `leader` and `follower`, both of initial size zero:

```
semaphore leader(0);
semaphore follower(0);
process p1 {
   dance() asa (leader.wait(p1)
      asa follower.signal());
}
process p2 {
   dance() asa (follower.wait(p2)
      asa leader.signal());
}
```

If the leader arrives before follower, the leader signals the follower which increments follower's count and leader goes to sleep as soon as follower signal is successful since leader count is zero. When a follower arrives, the follower signals the leader which wakes up the leader. The follower counter is already greater than zero, so the follower proceeds to dance. The leader wakes up, checks the counter to be greater than zero, decrements it and also proceeds to dance.

The mutex resource structure is identical to semaphores with the difference that mutex private counter value is always zero or one.

Stack Resource: A stack, also called a LIFO (last-in-first-out), resource in RealSpec is defined by following schema. The stack resource is multi-thread safe by using internal semaphores and mutex resources.

```
stack(int value, int pol=FCFS_BASED)
{
   list slist = [];
   semaphore available(size, pol);
   semaphore occupied(0, pol);
   mutex m(pol);
   int size = value;
   generic operator<<(generic x, generic p);
   generic operator>>(generic p);
   int items();
   bool full();
   bool empty();
   generic top();
}
```

The stack is internally modeled using a list variable `slist`. Two semaphores `available` and `occupied` are used for controlling read from empty stack and write to a full stack. A mutex resource `m` is used to lock and unlock stack access for a thread. The write operator (<<) adds an item `x` to the top of the stack as soon as the thread `p` writing to the stack locks the stack. The calling thread is put to sleep within the available semaphore if the stack is full. Likewise, the calling thread is put to sleep within the mutex `m` if the stack is locked by another thread. The write operator returns the updated stack. The read operator (>>) works in a similar manner and returns the item at the top of the stack. Consider following specification example using stack for evaluating a simple arithmetic expression written in postfix notation. The **`is current`** declaration is used for nested iteration by "freezing" each individual expression in the input stream into the variable `in` for parsing and evaluation.

```
stack s(10);
evaluate(input) = calc(s, in)
   where {
      in is current input;
      calc(s, n) = if n == nil
         then s >> result
         elseif isdigit(hd(n))
         then calc(s << hd(n),tl(n))
         else calc(op(s, hd(n)), tl(n));
      op(s, c) = case c of {
```

```
            '+': s << (s >> two + (s >> one));
            '-': s << (s >> two - (s >> one));
            '*': s << (s >> two * (s >> one));
            '/': s << (s >> two / (s >> one));
         }
      }
```

Queue Resource: A queue resource in RealSpec is bounded to a fixed number of items meaning that it has a specific capacity. Moreover, like stack, the queue resource is multi-thread safe by using internal semaphores and mutex resources. The queue resource has a similar schema as the stack. The operations of the write (<<) and read (>>) operators are different such that << writes to the end of the queue and >> reads from the front of the queue.

Array Resource: RealSpec provides single dimension array resource. The array resource is multi-thread safe by using internal mutex resource. The array resource is defined by following schema:

```
array(int value, int pol=FCFS_BASED)
{
   list alist = [];
   mutex m(pol);
   int size = value;
   generic operator<<(generic x, generic p);
   generic operator>>(generic p);
   generic operator[](int index);
}
```

The write (<<) and read (>>) operators work in the same manner as in stack and queue except that the write or read is performed at the array location identified by the index operator []. The following example uses arrays to perform selection sort of a stream of numbers sequences. The example assumes that the input sequences have ten or less numbers. The example also assumes that the input sequence has already been read into the array `a`.

```
array a(10);
ssort(a, items) = a asa k == 0
   where {
      A is current a;
      k = items-1 fby k-1;
      a = A fby swap(a, imax, k);
      imax = imax asa j == K+1
         where {
            K is current k;
            A is current a;
            j = 1 fby j+1;
            imax = 0 fby
               if (A[j]>>a) > (A[imax]>>b)
               then j else imax;
         }
      swap(a, imax, k) = …
   }
```

6. Dining Philosophers in RealSpec

Dijkstra invented the *dining philosophers problem* as a synchronization problem [10]. Imagine that five philosophers sit around a circular table with a plate of spaghettis. However, there are only five forks available. Each philosopher thinks. When a philosopher gets hungry, the philosopher sits down and picks up the two forks that are closest to the philosopher. If a philosopher can pick up *both* forks, the philosopher eats for a while. After a philosopher finishes eating, the philosopher puts down the forks and starts to think. The classic problem is used as a specification example for RealSpec so focus can be directed on the specification in RealSpec rather than the problem itself. Also, it is assumed that the duration between picking up left and right forks is zero avoiding deadlock.

The problem is specified in RealSpec by declaring five forks as **`mutex`** resources shared between five philosopher process threads `p1` to `p5`. The five forks are defined as a list constant by enclosing within the [% and %] brackets. A philosopher process is defined that consists of five threads with eat and think functions. When a process is created, all defined threads are automatically created and start simultaneous execution. Each philosopher represented by the five threads p_i is defined by the value of the function `eatandthink()`. The function `eatandthink()` is recursively defined in the **`where`** clause as an equation with initial value of `eat` followed by the value of `think` which in turn is followed by the call to `eatandthink()` to represent repeatability of eating and thinking for each philosopher. A philosopher starts eating as soon as the philosopher has forks and the duration of the eating is specified using temporal operator **`@dur`**. The philosopher tries to lock two adjacent forks represented by two adjacent mutex from the `forklist`. If both forks are locked, the philosopher starts eating; otherwise the thread is suspended until both forks are available. The `think` function works in the similar manner but does the opposite things. The function unlocks the right fork as soon as the left fork is unlocked followed by thinking for duration `thinktime`.

7. Summary and Future Work

The paper presented RealSpec executable specification language in general and resource modeling concepts in in particular. RealSpec real-time specification language is based on the functional, declarative, dataflow programming language Lucid. The abstract data resources were discussed in detail and the resource concepts were applied to specify the dining philosophers' problem. A language compiler is currently under implementation to be able to execute and debug specifications of real-time systems in

RealSpec by extending existing Lucid compiler.

Acknowledgment

This material was based on work supported by the National Science Foundation, while working at the Foundation. Any opinion, finding, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

References

[1] A. A. Khwaja and J. E. Urban, "A Framework for the Evaluation of Real-Time Specification Techniques," *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, Vol. 16, No. 6, December 2006, pp. 987-1014.

[2] I. Lee, P. Bremond-Gregoire, R. Gerber, "A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems," *Proceedings of the IEEE*, Vol. 82, No. 1, January 1994, pp. 158-171.

[3] A. A. Khwaja and J. E. Urban, "Comparison of Real-Time Specification Techniques Using a Real-Time Features Based Framework," *Proceedings of the 10th IASTED International Conference on Software Engineering and Applications*, November 13-15, 2006, Dallas, TX, USA, pp. 509-516.

[4] D. K. Hammer and M. R. V. Chaudron, "Component-Based Software Engineering for Resource-Constraint Systems: What are the Needs?" *Proceedings of the IEEE 6th International Workshop on Object-Oriented Real-Time Dependable Systems*, January 8-10, 2001, Rome, Italy, pp. 91-94.

[5] J. A. Stankovic, "Real-time and Embedded Systems," *ACM Computing Surveys*, Vol. 28, No. 1, March 1996, pp. 205-208.

[6] W. W. Wadge and E. A. Ashcroft, *Lucid – The Dataflow Programming Language*, Academic Press, London, 1985.

[7] P. J. Landin, "The Next 700 Programming Languages," *Communications of the ACM*, Vol. 9, No. 3, March 1966, pp. 157-166.

[8] E. A. Ashcroft and W. W. Wadge, "Lucid – A Formal System for Writing and Proving Programs," *SIAM Journal of Computing*, Vol. 5, No. 3, September 1976, pp. 336-354.

[9] B. Freeman-Benson, "Lobjcid: Objects in Lucid," *Proceedings of the 4th International Symposium on Lucid and Intensional Programming*, April 29-30, 1991, SRI International, Menlo Park, California, USA, pp. 80-87.

[10] E. W. Dijkstra, "Hierarchical Ordering of Sequential Processes," *Acta Informatica*, Vol. 1, No. 2, June 1971, pp. 115-138.

```
system {
    resources {
        mutex f1, f2, f3, f4, f5;
    }
    processes {
        philosopher threads { p1; p2; p3; p4; p5; };
    }
}

process philosopher thread p1, p2, p3, p4, p5 {
    eatandthink()
        where {
            eatandthink() = eat() fby think() fby eatandthink();

            eat() = (eating asa hasforks) @dur eattime m
                where {
                    eating = 'Philosopher ' ^ mkstring(pid) ^ ' is eating!';
                    hasforks = leftfork.lock() asa rightfork.lock();
                }

            think() = (thinking asa putforks) @dur thinktime m
                where {
                    thinking = 'Philosopher ' ^ mkstring(pid) ^ ' is thinking!';
                    putforks = rightfork.unlock() asa leftfork.unlock());
                }

            forklist = [% f1, f2, f3, f4, f5 %];
            rightfork = ith(pid, forklist);
            leftfork = ith((pid+1)%5, forklist);
            eattime = 5;
            thinktime = 10;
        }
}
```

Figure 1. Dining Philosophers Specification

PREDICTING CHANGE PROPAGATION IN OBJECT-ORIENTED SYSTEMS: A CONTROL-CALL PATH BASED APPROACH AND ASSOCIATED TOOL

LINDA BADRI, MOURAD BADRI & DANIEL ST-YVES

Software Engineering Research Laboratory
Department of Mathematics and Computer Science
University of Quebec at Trois-Rivieres
C.P. 500, Trois-Rivières, Québec, Canada, G9A 5H7
{Linda.Badri | Mourad.Badri | Daniel.St-Yves}@uqtr.ca

Abstract

Change impact analysis plays an important role in software maintenance. It allows evaluating the possible effects of a change. We present, in this paper, a static technique supporting predictive change impact analysis for object-oriented systems and associated tool. The technique uses a model based on control-call flow graphs obtained by static analysis. The control-call flow paths, generated in a compacted form from the model, are used to predict change propagation. The technique we present has been compared, according to several criteria (quality of precision and performance), to two static impact analysis techniques (call graph and slicing based techniques). We used several versions of a Java project (JMOL). The results are reported and discussed in the paper. We also give an overview of the environment we developed to support the proposed approach.

1. Introduction

Maintenance is widely recognized as a very costly step of the software development process [2, 3]. Two basic activities of software maintenance are [1]: the comprehension of the system and the evaluation of the effects of a change. To understand the potential effects of a change, a developer must first understand how the system works [4]. To reduce the important costs of software maintenance, it is necessary to ease the change management process using approaches that allow, among others, evaluating the potential effects of a change [5, 6, 7, 8, 9]. Furthermore, the complexity of interdependencies between components of a software makes this task even difficult [10, 11, 12]. A change to a software, however small, can lead to unexpected ripple effects [13].

Change impact analysis, often called *Impact Analysis*, plays a crucial role in this context. It allows developers and managers to evaluate the possible effects of a change to the source code of a program [15, 16]. Impact analysis information can be used to plan changes, to execute changes, as well as to follow the effects of a change [14]. One of the main goals is to insure, in an iterative process, the consistency of a system before and after a change was implemented [17, 18, 19]. Impact analysis is often used to evaluate the effects of a change after it was implemented [13]. However, more proactive approaches could use impact analysis techniques before it is implemented [5, 13]. In this way, they would allow developers assessing and choosing, among several ways of implementing a change, the solution presenting the lowest estimated impact. Predictive approaches give a global overview of the required efforts in terms of costs and planning [4, 13]. Barros et al. [1], among other authors, discussed the importance of determining the effects of a planned change before it is implemented. The predictive analysis of an impact requires, however, that the maintainers specify the approximate location where the planned changes will be implemented. Impact analysis techniques can be divided into two major classes [5]: traceability and dependence analysis techniques. Impact analysis techniques can be dynamic such as [13, 14, 24, 26], static such as [4, 10, 21, 22, 27, 29], or techniques combining both static and dynamic analysis such as [23]. We focused in this paper on static dependence analysis techniques.

We present a static technique supporting predictive change impact analysis for object-oriented systems and associated tool. The technique uses a model based on control-call flow graphs (a reduced form of control flow graphs) obtained by static analysis of the source code. The atomic instructions of a control flow graph that do not contain method calls are ignored. The control-call flow paths, generated in a compacted form from the model, are used to predict change propagation. Working at the method level granularity makes the analysis more appropriate in practice [13]. We conducted an empirical study using several versions of a Java large project (JMOL). The observed changes were collected from its successive versions. The technique we present has been compared, according to several criteria (quality of precision and performance), to two static impact analysis techniques (call graph and slicing based techniques). The sets of potentially affected classes, returned by the three techniques after a given change, were compared to the observed changes. The results are reported and discussed in Section 5.

The remainder of the paper is organized as follows. Section 2 presents an overview of call graph and slicing based impact analysis techniques. The impact analysis technique we propose is presented in Section 3. Section 4 gives a brief overview of the developed tool. Section 5 presents the empirical study we conducted and discusses the obtained results. Finally, we give some conclusions and future work directions in Section 6.

2. Related work

Impact analysis based on call graphs [21] considers when a procedure P is modified, all the procedures called by P (directly or indirectly) as potentially affected. Impact analysis, in this way, can lead to imprecise results [13, 23]. In fact, we cannot determine, from a simple call graph of a procedure P, what are the conditions related to the propagation of the impact of a change from a procedure to the other procedures. Moreover, call graphs cannot capture the propagation of an impact resulting from the return to a procedure [13]. This weakness was also discussed in [23]. Call graphs capture the local structure of potential calls. They ignore, however, the other aspects related to their control. The behavior of calls is much more complex that call graphs indicate. The first experimentation we conducted and discussed in [27], which focused on a partial comparison between the proposed technique and call graphs based techniques, confirmed the conclusions of several other researches. It was limited, however, to an analysis of the size of the returned impact sets and has not assessed their quality in terms of precision. We implemented, for the needs of our experimentation, two versions of call graph based impact analysis (direct and indirect).

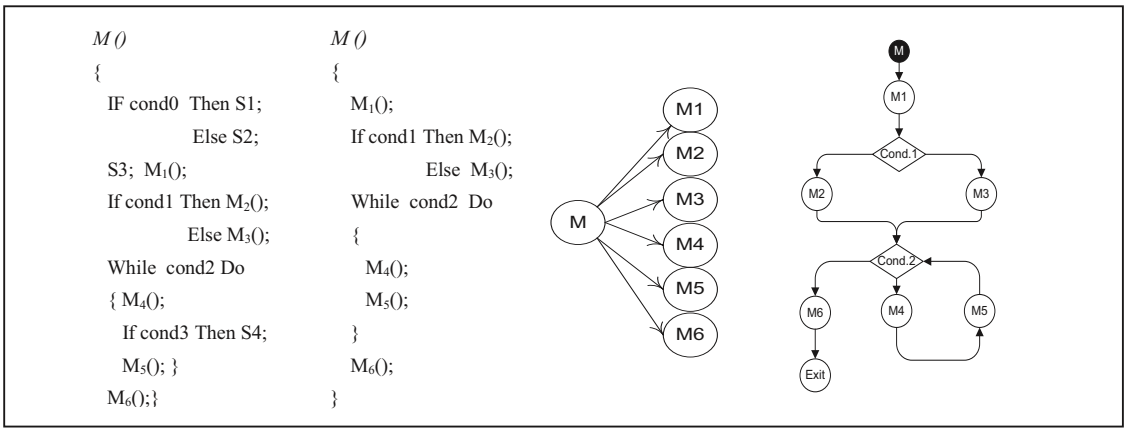
Program slicing was introduced by Weiser in [30] to support debugging and program comprehension. The proposed algorithm is based on an analysis of data flow in a control flow graph. It determines the parts of a program that are potentially affected by the values of the variables of a slice criterion [35, 18, 36]. Two sets of data are defined: defined variables and referenced variables, for each node (instruction) of the graph [31]. The Weiser algorithm was adapted by several authors [32, 33, 29]. Bishop [34] focused on the improvement of the performance (time) of impact analysis using traditional slicing techniques. Others authors [18, 13, 14] assessed the performance of static forward slicing and particularly the size of the returned sets of potentially affected parts of a program after a change. Wang et al [18] discussed the role of program slicing in a ripple-effect process. They implemented a prototype for the analysis of Cobol programs using backward and forward slicing. They concluded, after several experimentations, that the indirect propagation returns a too large set of potentially affected elements compared to a direct propagation. Furthermore, according to [18], it is preferable to analyze iteratively the direct results when performing a change process. However, Wang et al. [18] did not explore the quality of precision of the returned results. They did not precise whether the direct results are better, in terms of quality of precision, compared to those obtained indirectly. Law et al. in [13] conclude that static slicing can be costly in resources and can return imprecise results when dynamic behaviors are analyzed. Moreover, program slicing is difficult to implement [37]. An implementation of forward slicing based on the approach of Weiser [30] was realized in our work. We implemented, in fact, a version of static forward slicing similar to the definition of *Data Slicing* of Zhang et al. [38]. The algorithm we implemented focuses on the identification of methods containing one or several paths of impacted data. Starting from a control flow graph, we adjust the intra and inter modular data dependence links between the different atomic instructions (irreducible instructions) of a method.

3. Control-call flow graph based impact analysis

Definition 1 : A control flow graph (CFG) is a directed graph. The nodes of the graph represent decision points (if-then-else, while, case, etc.), an instruction or a sequential bloc of instructions. A sequential bloc of instructions S is a sequence of instructions such that if we execute the first instruction, we are sure to execute the others, and always in the same way. A directed arc links node N_i to node N_j if it is possible to execute the instruction corresponding to N_j immediately after the one associated to node N_i. The arcs of the graph indicate the control from one node to another.

Definition 2 : A control-call flow graph (CCG) is a control flow graph from which the nodes representing instructions not leading to calls are removed.

Let us consider the example of method M given in figure 1.1. S_i represents a sequence of instructions that do not contain method calls. Figure 1.2 gives the body of method M reduced to calls. The corresponding call graph is given in figure 1.3. Finally, figure 1.4 gives the corresponding control-call flow graph. The proposed technique is organized in several steps.



(1.1) (1.2) (1.3) (1.4)

Fig. 1. Portion of a method, its reduced form and corresponding call graph and control-call graph.

```
M ()                          M2()                        M6()
{  M1();                      {  M7();                    {  If cond4 Then M8();
   If cond1 Then M2();              If cond3 Alors M8();        M10();
            Else M3();        }                           }
   While  cond2  Do           M3()                        M8()
   {                          {                           {
     M4();                      M8();                       M9();
     M5();                    }                           }
   }
   M6(); }
```

Fig. 2. Synthesis (reduced form) of several methods.

1. $M : M_1 (M_2 / M_3) \{ M_4, M_5 \} M_6$
2. $M_2 : M_7 [M_8]$
3. $M_3 : M_8$
4. $M_6 : [M_8] M_{10}$
5. $M_8 : M_9$

Fig.3. Compacted control-call paths.

$M, M_1, M_2, M_7, M_8, M_9, M_6, M_8, M_9, M_{10}$

$M, M_1, M_2, M_7, M_8, M_9, M_6, M_{10}$

$M, M_1, M_3, M_8, M_9, M_6, M_{10}$

$M, M_1, M_2, M_7, M_8, M_9, M_4, M_5, M_6, M_8, M_9, M_{10}$

$M, M_1, M_3, M_8, M_9, M_4, M_5, M_6, M_8, M_9, M_{10}$

$M, M_1, M_2, M_7, M_8, M_9, M_4, M_5, M_4, M_5, M_6, M_8, M_9, M_{10}$

……..

Fig. 4. Some control-call paths.

Step 1. Construction of control-call flow graphs: The control-call flow graphs, corresponding to the different methods of the program, are extracted by static analysis of the source code.

Step 2. Generation of compacted control-call paths: From the control-call flow graphs, we generate the control-call paths in a compacted form. Starting from the compacted paths, we can generate the set of reduced control-call paths by eliminating unfeasible ones. The compacted paths allow informing on the dynamic behavior of a program. This represents an advantage relatively to dynamic approaches that attempt to obtain these paths by analyzing and compacting execution traces. Figure 2 gives the synthesis (control reduced to calls) of several methods that we consider to illustrate our approach. Figure 3 presents the compacted control-call paths corresponding to the methods of figure 2. We use several notations to express the control in the call sequences. The notation {*sequence*} expresses the iteration in the execution of the sequence (or part of a sequence). The sequence in { } can be executed 0 or several times. The sequence (*sequence 1* / *sequence 2*) expresses an alternative in the execution of the two sequences. The sequence [*sequence*] expresses the fact that the sequence between [ ] can be executed or not. Figure 4 illustrates some of control-call paths that can be deduced from figure 3.

Step 3. Impact Analysis: Our technique considers, if we plan a change on a method M, solely the impact that can be propagated through any control-call flow path including method M [27]. This principle was already used in [13]. Any method called after M, called by M, and any method calling M is included in the set of potentially affected methods.

4. Iterative Impact Analysis: A Supporting Tool

We developed a tool (JIAT: Java Impact Analysis Tool) as a plug-in for the Eclipse development environment. Others authors have also implemented their impact analysis tool using the Eclipse API [23, 39]. It allows, in fact, the creation of *Abstract Syntax Tree* (AST), which facilitates the analysis of the code of a project. Their use makes also the tool independent from the version of the Java language. The architecture of the developed tool is given in figure 5. The tool compiles, using Eclipse environment, a set of AST for a given project. Then, the CCG engine converts the AST in standard CCG structures for each method of the project. Finally, the analysis engine uses the set of CCG to generate the control-call flow paths and capture the potentially affected classes (methods). The analysis process is done in three steps:

Change location: The tool allows the user to indicate the location of the planned changes. This is done by positioning the cursor in a method from the Eclipse's Java file editor. An example is given in figure 6.

Analysis of change propagation: The user asks the tool to do a change impact analysis from the position of the cursor. The tool compiles the CCG and gives different views of the results.

View of the results: The results are presented in two different forms: textual and graphic. The textual view can be interpreted as a list of affected classes and methods (figure 7). Figure 7 shows two windows: the first one shows the class or the method that will be modified and the second one shows the set of classes (methods) that could be impacted. The graphic view, shown in figure 8, allows the user to have a global view of the extent of the propagation of the change.

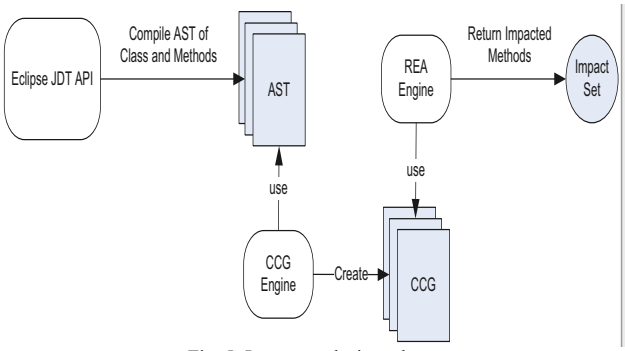


Fig. 5. Impact analysis tool.

```
public static BaseAtomType get(int atomicNumber) {

  Enumeration iter = typePool.elements();
  while (iter.hasMoreElements()) {
    BaseAtomType at = (BaseAtomType) iter.nextElement();
    if (atomicNumber == at.getAtomicNumber()) {
      BaseAtomType atr = get(at.getRoot());
      return atr;
    }
  }
  return null;
}
```

6. Location of a change.

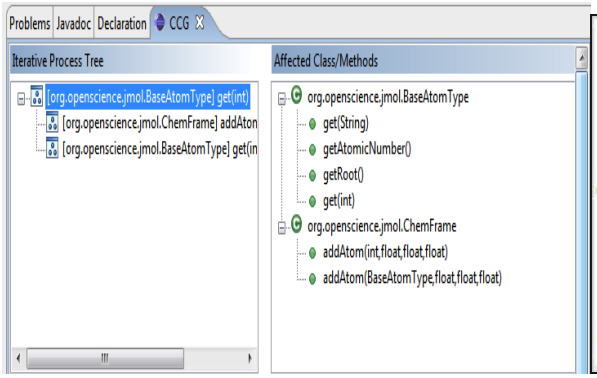


Fig. 7. Presentation of the results in textual form.

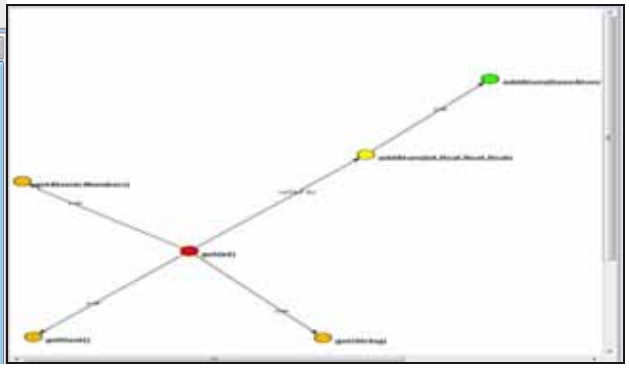


Fig. 8. Presentation of the results in graphic form.

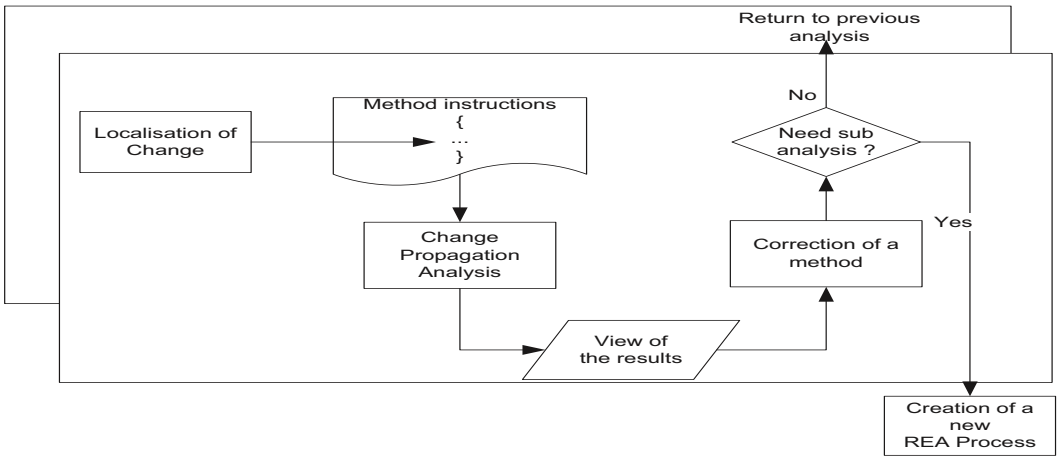


Fig. 9. Iterative process for the construction of impacted sets.

The visualization of the propagation can be done, according to the choice of the user, in two different ways: the first shows the parts of the code that are directly affected, and the second shows the parts that may be affected indirectly. The user can select a class (method) in the list of affected classes (methods) (figure 7, window 2) and restart the analysis process. He can also return to a previous state by selecting another class (method) in window 1. We consider, for simplification reasons, solely the set of elements (classes, methods) that are potentially affected directly following a given change. The set of impacted elements is constructed iteratively (figure 9). In this way, the technique avoids managing eventual large sets that can contain several non impacted elements. In the process, only the elements that can be affected directly following, for example, a modification to a method M_i are considered. Let us suppose that this set is noted I_{Mi}. If after the modification of M_i, another method M_j belonging to I_{Mi} needs an adaptation, we can repeat the process that consists on constructing the corresponding I_{Mj} set. This process is repeated iteratively until there are no more impacted methods by the change. In this way, we avoid considering, for regression testing, a relatively large set of methods that do not actually need to be tested again.

5. Case Study

5.1. Methodology and data collection

We conducted an empirical study using several generations of an open-source Java project (*JMOL*, available on www.*sourceforge.net*). JMOL is a free application for the visualization of molecules for students, educators and researchers in chemistry and biochemistry. The project represents a concrete case study and was used, among others, in [40]. The goal is, in particular, to determine the precision of the considered approaches in identifying the right impacted classes (methods) after a change. In a first step, we identified between two successive versions of JMOL the different instantiated changes. The observed changes from one version to another of JMOL were collected. We applied the selected approaches: Call Graph (CG), Control-Call Graph (CCG) and Forward Slicing (FS). The sets of potentially affected classes (methods), returned by the three techniques, were compared to the observed changes. The three techniques were also compared using some performance criteria. Our experiments followed a methodology similar to the one used in [25]. We extended it by assessing the performances of the three approaches.

Identifying changes and impacted classes: We compared (repeatedly for each version), using the binary comparison tool of Eclipse, two major successive versions of JMOL to identify the methods (classes) changed from one version to the other. This allowed us to extract, for each version, the set of changed methods (classes). Moreover, when a programmer makes a change (initial change) on a method for example, it is possible that other methods will be modified (adapted) as a consequence of the initial change. These changes (propagation) must be identified. To identify the initial changes, we combined the use of the binary comparison tool of Eclipse with a manual analysis of the code. This task was not easy to perform.

Filtering the elements that do not induce propagation: To construct adequately the set of elements that were really affected, from one version to another, it is important to filter the elements that do not generate a propagation of change such as it is mentioned in [25]. Several changed methods where ignored according to some criteria such as: reformatting the code (adding spaces, empty lines), changing informative elements (changing comments), etc.

5.2. Used metrics

Hassan et al. [25] defined two metrics to compare ripple-effects analysis approaches. They proposed the metric *recall* defined as the percentage of the number of elements identified by an approach (*PO*) on the number of elements that were really changed. This measure indicates the sensitivity of the approach. For example, a *recall* of 0.3 signifies that the approach has correctly identified for example 30% of the modified classes, but 70% of the classes really modified were not identified. They proposed also the *precision* measure defined as the percentage of elements correctly identified by the approach (*PO*) on the total number of identified elements (*P*). For example, a *precision* of 0.5 implies that for two classes returned by an approach, one of the classes is not really impacted by a change.

5.3. Results and discussion

Collecting and analyzing data on changes from JMOL for the assessment of three approaches was not easy. Our experiments have been limited to the seven first versions of the software, which presently count about 10 versions. The collected data were significant enough to allow us to complete our experiments. Table 1 gives some descriptive statistics on the different versions of the project. We can observe, among other things, that the size (number of lines of code) has gone from 18357 to 30288. This represents an increase of 65%. The number of classes was 237 and is now 315 (33%). The number of methods has gone from 1073 to 1732 (61%). The transition from version 1 to 2 was not retained, because there was not enough data.

Table 2 summarizes the obtained results. The three evaluated approaches (implemented for Java programs) are shown in the table: CG_i (indirect call graphs), CG_d (direct call graphs), CCG (control-call flow graphs) and FS (forward slicing). Column *C* indicates the number of changed classes between two versions. The total number of observed changes is 121. Column *P* notes the total number of classes that each approach returned and column *PO* notes the classes that were correctly identified by each approach. CCG obtains a better sensitivity with a *recall* of 32%, higher than FS (22%), and CG_d (30%). However, FS obtains a *precision* of 58% where CCG obtains 48%. CG_i has the lowest *precision*. CCG gives better results than CG_d by having a better *precision* and *recall*. Compared to FS, CCG is less restrictive in its analysis.

Table 1. Descriptive statistics on the different versions of project *Jmol*.

Versions	# att. pub.	# att. prot.	# att. priv.	# classes	# méthodes	LOC
1	100	47	569	237	1073	18357
1.1	155	48	585	256	1153	20228
1.2	155	48	588	259	1164	22548
2	65	35	575	206	950	18166
3	64	52	590	214	996	18666
4	59	54	535	225	956	19324
5	59	54	589	249	1079	21908
6	181	44	529	293	1656	28252
7	183	49	541	313	1718	29650
8	182	49	553	315	1732	30288

Table 2. **"**Recall" and "Precision" of the three approaches.

REV	OCCURRED	CGi				CGd				CCG				FS			
	C	P	PO	Recall	Prec.	P	PO	Recall	Prec.	P	PO	Recall	Prec.	P	PO	Recall	Prec.
2-3	15	64	11	0.73	0.17	15	7	0.47	0.47	23	7	0.47	0.30	6	5	0.33	0.83
3-4	20	42	10	0.50	0.24	15	4	0.20	0.27	5	3	0.15	0.60	4	2	0.10	0.50
4-5	14	25	7	0.50	0.28	10	6	0.43	0.60	12	7	0.50	0.58	16	6	0.43	0.38
5-6	43	50	18	0.42	0.36	19	10	0.23	0.53	19	10	0.23	0.53	43	8	0.19	0.19
6-7	29	17	8	0.28	0.47	17	5	0.17	0.29	18	7	0.24	0.39	1	1	0.03	1.00
	121	198	54	0.49	0.30	76	32	0.30	0.43	77	34	0.32	0.48	70	22	0.22	0.58
				average				average				average				average	

Table 3. Performances of the three approaches.

REV	CGi			CGd			CCG			FS		
	AST	Analysis	Mem.	AST	Analysis	Mem.	AST	Analysis	Mem.	AST	Analysis	Mem.
2-3	1638	10795	49.8	1695	156	47.1	1671	140	41.2	1695	10584	48.5
3-4	2096	12630	62.5	1971	174	50.1	1732	171	44.2	1826	10888	45.3
4-5	1834	11369	58.9	1957	130	52.6	1763	234	39.3	1544	11325	44.2
5-6	1988	16850	72.6	2011	195	59.1	2044	187	45.0	2054	16021	51.2
6-7	2808	46473	94.4	2305	265	75.0	2627	152	75.3	2614	27308	63.6
avg.	2073	19623	67.6	1988	184	56.8	1967	177	49.0	1947	15225	50.6

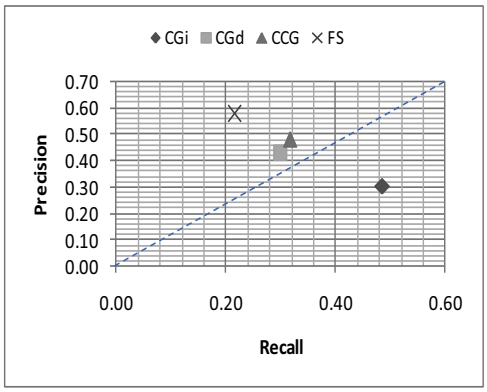


Diagram 1. "Recall" vs "Precision": class level.

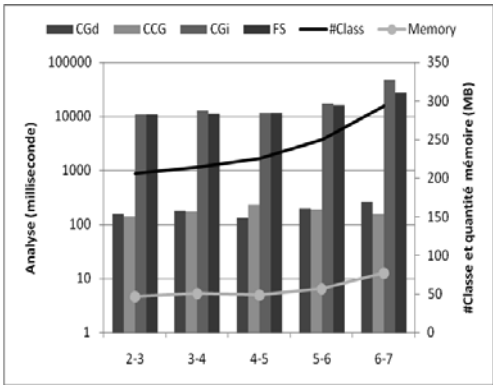


Diagram 2. Analysis time compared to the number of classes and quantity of memory used by revision.

Moreover, Zhang et al. report in [38] that they observe a very short distance (in instruction granularity) of the impact propagation by doing a dynamic FS. We also observed a small distance for static FS. Diagram 1 gives a global view on the sensitivity and precision of the approaches. The more an approach follows the diagonal line, the more its results are balanced (*sensitivity* and *precision*). The more the approach is close to the higher right corner, the more it is effective. FS obtains a high *precision* compared to the other approaches, but obtains a low *recall*. CG_i obtains the best *recall.* It also obtains the worst *precision*. CG_d obtains a better precision compared to CG_i, but a low sensitivity. CCG seems to be more effective than the other techniques. It gives results more balanced than the other approaches (sensitivity and precision - closest to the top right corner of the diagram).

Table 3 reports on time, in milliseconds, for the creation of an Eclipse AST structure for each approach, as well as analysis time, which include the creation of their respective dependence structures. The use of memory of each approach is also noted for each experiment. From the reported results, we can observe that on average the creation of an AST takes 2 seconds. Both direct approaches obtain the fastest results on analysis, which is approximately 1/10 of seconds and is relatively fast. For CG_i and FS techniques, the analysis time is very imposing. On average, they both take 19 and 15 seconds respectively to give a result. To do 5 analyses CG_i and FS take more than a minute, while CG_d and CCG techniques take less than one second. The quantity of memory used is similar in all approaches. This is due to the intensive use of Eclipse's AST to implement the approaches. CG_i seems to be using more memory than the others. The creation of matrices to determine the indirect relations could explain a higher use of memory. In diagram 2, we show also the required time (left axis). Visually, it seems that there exists a link between the increasing number of classes from one revision to another, the required time to complete an analysis, and the average quantity of memory used for all approaches. The more there is information to analyze, the

more the memory required by each approach increases. This remark is also true for the required time. Furthermore, the direct approaches are largely faster than the indirect approaches and seem more or less influenced by the increased number of classes between revisions.

6. Conclusions and future work directions

We presented, in this paper, a static technique supporting predictive change impact analysis for object-oriented systems and associated tool. The technique uses a model based on reduced control flow graphs (control-call flow graphs) obtained by static analysis of the source code. We compared it according to several criteria (quality of precision and performance) to two static impact analysis techniques (call graph and slicing based techniques).

We performed an empirical study using several versions of a Java large project (JMOL). The obtained results show that CCG and FS techniques are more precise than CG based approaches. Moreover, CCG obtains a better sensitivity than FS. However, FS obtains a better precision than CCG. CCG seems to be more effective than the other techniques. It gives results more balanced than the other approaches (sensitivity and precision - closest to the top right corner of the diagram). On the performance level (analysis time), CCG is much faster than FS. In a general manner, and taking into account the criteria used for the assessment of the techniques, CCG technique seems presenting a good compromise relatively to the other techniques. The present study has also confirmed the relevance of taking into consideration the control flow and data flow in impact analysis. The unification of the two approaches would be an interesting way to explore.

Moreover, the tool supporting our approach can be used to evaluate, by successive simulations of changes, their estimated impact sets. This possibility allows choosing, among several solutions for implementing a given change, the one having the lowest estimated impact. This presents an advantage compared to the FS approaches which are applied generally after a given change is implemented. As future work, we plan to: (1) Extend our technique to take into account data flow, (2) and experiment the new model.

Acknowledgments

This project was financially supported by NSERC (National Sciences an Engineering Research Council of Canada).

References

[1] S. Barros, Th. Bodhun, A. Escudie, J.P. Voidrot., Supporting Impact Analysis : A semi automated technique and associated tool. In Proc. of the 1995 IEEE Conf. on Software Maintenance, pp. 42-51, Piscataway, NJ, 1995.

[2] Ian Sommerville, Software Engineering, Seventh edition, Pearson, Addison Wesley, 2004.

[3] R.S. Pressman. Software Enginering: A partionner'a approach, 6th edition, Mc Graw Hill, 2005.

[4] Michelle Lee, A. Jefferson Offutt and Roger T. Alexander. Algorithmic Analysis of the Impacts of Changes to Object-Oriented Software. IEEE, pp. 61-70, 2000.

[5] S.A. Bohner and R. Arnold. Software Change Impact Analysis. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[6] J. Han. Supporting Impact Analysis and Change Propagation in Software Engineering Environments, In Proceedings of the 8th Intl. Workshop on Software Technology and Engineering Practice (STEP'97), London, England, pp. 172-182, July 1997.

[7] V. Rajlich, Modeling software evolution by evolving interoperation graphs. Ann. Softw. Eng. 9, 1-4, 235-248, Jan. 2000.

[8] Black, S. 2001. Computing ripple effect for software maintenance. Journal of Software Maintenance 13, 4, 263, 2001.

[9] Sarita Basil and Rudolf K. Keller. Software Visualization Tools: Survey and Analysis, 2001.

[10] Kung, D. C., Gao, J., Hsia, P., Wen, F., Toyoshima, Y., and Chen, C., Change Impact identification in object-oriented software maintenance. In Proc. of the International Conf. on Software Maintenance, pp. 202-211, 1994.

[11] Richard J.Turver and Munro Malcom, An early impact analysis technique for software maintenance. Journal of Software Maintenance, Research and Practice, 18(12):35-52, January-February 1994.

[12] L. Li and A. J. Offutt. Algorithmic analysis of the impact of changes to object-oriented software. In Proc. of the IEEE International Conf. on Software Maintenance, CA, USA, pp 171-184, 1996.

[13] J. Law, G. Rothermel. Whole Program Path-Based Dynamic Impact Analysis. In Proc. of the International Conf. on Software Engineering, pp. 308-318, 2003.

[14] A. Orso, T. Apiwattanapong, and M.J. Harrold. Leveraging field data for impact analysis and regression testing. In Proc. of European Software Engineering Conf. And ACM SIGSOFT Symp. On the foundations of software Engineering (ESEC/FSE'03), Helsinki, Finland, Sept. 2003.

[15] Barbara G. Ryder and Frank Tip. Change Impact Analysis for object-Oriented Programs. In ACM SIGPLAN-SIGSOFT Worshop on Program Analysis for Software Tools and Engineering, pages 46-53. ACM Press, 2001.

[16] M. Ajmal Chaumun, Hind Kabaili, Rudolf K. Keller and F. Lustman. A change impact model for changeability assessment in object-oriented software systems, Science of Computer Programming, Volume 45, Issues 2-3, , Pages 155-174, November-December 2002.

[17] Yau, S.S.; Collofello, J.S.; MacGregor, T., "Ripple effect analysis of software maintenance," Computer Software and Applications Conference, 1978. COMPSAC '78. The IEEE Computer Society's Second International , vol., no.pp. 60- 65, 1978.

[18] Yamin Wang and Wei-Tek Tsai and Xiaoping Chen and Sanjai Rayadurgam 1996. The Role of Program Slicing in Ripple Effect Analysis, In Proc of SEKE'96, p. 369-376, 1996.

[19] Haider Bilal, S Black. Using the Ripple Effect to Measure Software Quality, Software Quality Management-International Conference, 2005.

[20] Wei Li and Sallie Henry. Maintenance support for object-oriented programs. Journal of Software Maintenance, Research and Practice, 7(2):131-147, March-April 1995.

[21] S.A. Bohner. Impact Analysis in the Software Change Process : A Year 2002 Perspective. In Proc. of the International Conf. on Software Maintenance, 4-8 Nov. 1996.

[22] Briand, L.C., Wust, J., Lounis, H., Using coupling measurement for impact analysis in object-oriented systems. Proc. of the IEEE International Conf. on Software Maintenace (ICSM '99), 30 Aug.-3 Sept. 1999, pp.475 – 482.

[23] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara G. Ryder, and Ophelia Chesley. Chianti: A Tool for Change Impact Analysis of Java Programs. OOPSLA'04, Vancouver, British Columbia, Canada, Oct. 24-28, 2004.

[24] A. Orso, T. Apiwattanapong, J.Law, G. Rothermel, and M.J. Harrold. An Empirical Comparison of Dynamic Impact Analysis Algorithms. In Proc. of the International Conf. on Software Engineering (ICSE'04), , pp. 491-500, Edinburg, Scotland, 2004.

[25] Hassan, A.E.; Holt, R.C., Predicting change propagation in software systems, Proceedings. 20th IEEE International Conference on Software Maintenance, vol., no.pp. 284- 293, 11-14, Sept. 2004.

[26] B. Korel and J. Laski. Dynamic slicing in computer programs. Journal of Systems Software, 13(3): 187-195, 1990.

[27] Badri, L., Badri, M., and St-Yves, D. Supporting Predictive Change Impact Analysis: A Control Call Graph Based Technique. In Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05) - Volume 00 (December 15 - 17, 2005), IEEE Computer Society, 2005.

[28] S.S. Yau , J. S. Collofello. Some Stability Measures for software maintenance. IEEE Transactions on Software Engineering, 6(6): pp. 545-552, November 1980.

[29] S. Horwitz, T. Reps, and W. Binkley D. Interprocedural slicing using dependence graphs. ACM SIGPLAN Notices, Vol. 39, Issue 4, April 2004.

[30] M. Weiser. Program slices: formal, psychological, and pratical investigations of program abstraction method. PhD thesis, University of Michigan, Ann Arbor, 1979.

[31] Sebastian Danicic. Dataflow Minimal Slicing, PhD. Thesis, University of North London, 1999.

[32] L. M. Ottenstein and K. J. Ottenstein. The program dependence graph in software development environments. SIGPLAN Notices, Vol.19, pp. 177-184, 1984.

[33] J. Jiang, X. Zhou, and D. J. Robson. Program slicing for C, The problems in implementation. IEEE Inter. Conference on Software Maintenance, 1991.

[34] Luke Bishop. Incremental impact analysis for object-oriented software, Master Thesis, Iowa State University, 2004.

[35] F. Tip. A survey of program slicing techniques. Journal of Programming Language, vol. 3, pp.121-189, 1995.

[36] De Lucia, A. Program slicing: Methods and applications. In 1st IEEE International Workshop on Source Code Analysis and Manipulation (Florence, Italy, 2001), IEEE Computer Society Press, Los Alamitos, California, USA, pp. 142-149, 2001.

[37] K. B. Gallagher. Some Notes on Interprocedural Program Slicing. SCAM'04, Fourth IEEE International Workshop on Source Code Analysis and Manipulation, 2004.

[38] Zhang, X., Gupta, N., and Gupta, R., A study of effectiveness of dynamic slicing in locating real faults. Empirical Software. Eng. 12, 2 (Apr. 2007), 143-160, 2007.

[39] Jonathan Buckner, Joseph Buchta, Maksym Petrenko, Vaclav Rajlich. JRipples: A Tool for Program Comprehension during Incremental Change, IWPC, pp. 149-152, 13th International Workshop on Program Comprehension (IWPC'05), 2005.

[40] Tsantalis, N.; Chatzigeorgiou, A.; Stephanides, G., Predicting the probability of change in object-oriented systems, Software Engineering, IEEE Transactions on , vol.31, no.7pp. 601- 614, July 2005.

A Qualitative Assessment of the Reverse Engineering Capabilities of Unit Testing Tools for Understanding Java Programs

Andy Tinkham
Quality Assurance
GovDelivery, Inc.
andy@govdelivery.com

Scott Tilley
Department of Computer Sciences
Florida Institute of Technology
stilley@cs.fit.edu

Tauhida Parveen
Department of Computer Sciences
Florida Institute of Technology
tparveen@fit.edu

Abstract

The rise of agile methodologies and open source software has led to the creation of several new tools designed to help software engineers construct better applications. One particular emphasis of these new tools is the creation of unit tests. This paper describes a qualitative assessment of the reverse engineering capabilities of three unit testing tools (Agitar Software's Agitator, and the open source tools JUnit and EasyMock) for Java programs. The focus of the assessment is largely on how the unit testing tools can assist in understanding existing code. The assessment is structured using an existing Reverse Engineering Environment Framework (REEF). The results of the assessment suggest that unit testing tools can indeed aid software engineers in understanding Java programs.

Keywords: program understanding, reverse engineering, unit testing, qualitative, assessment, REEF, Java

1. Introduction

One area of software engineering tool functionality that has grown rapidly over the past few years is that of unit testing. This is due, in part, to the growing popularity of Agile methodologies such as Extreme Programming [9] or Scrum [12]. Many of these development methodologies place a strong emphasis on the developers creating robust unit test suites (often even before the code itself is written) and executing these tests frequently. Numerous tools, both commercial and open source are available for creating unit tests. While these tools have an established reputation for aiding in developing unit tests, there are other uses to which they can be put as well – in particular, they have inherent abilities to aid program understanding by reverse engineering.

This paper assesses the reverse engineering capabilities of unit testing tools from the perspective of a developer trying to gain understanding of an existing program (in particular Java program for this research). A reverse engineering framework REEF [14] is used to assess the capability of program understanding of these tools. The assessment is qualitative in the sense that the unit testing tools are not evaluated according to absolute ability, but rather according to the types of reverse engineering activities that they support. The unit testing tools that are considered in this paper are quite popular: Agitator (from Agitar Software) [6], Junit [3], and EasyMock [1].

The next section of the paper briefly describes background for this research. Section 3 describes the Reverse Engineering Environment Framework (REEF), the instrument that it used to structure the assessment. Section 4 describes the three unit testing tools examined as a part of this study and the results of the evaluation performed. Finally, Section 5 summarizes the paper and discusses possible avenues for future work.

2. Background

Unit tests examine individual pieces (or *units*) of a system. Units are generally classes or methods in a typical object-oriented program, and unit tests focus on the functionality contained within the unit. These tests simply check that the unit behaves in the manner it was designed. Unit tests are generally written by developers in conjunction with the code that comprises the unit being tested. Traditionally, developers have been reluctant to create unit tests, to the lament of the testers on those projects. Little tool support existed that was used in practice, and developers had no training or support materials for creating these tests.

The emergence of agile methodologies brought a change to all this. Most agile methodologies put a strong emphasis on creating and maintaining a suite of unit tests to serve as a "safety net" to ensure that making changes to the code (either to fix bugs or add new functionality) does not break existing functionality. Test Driven

Development (TDD) is one form of agile software development where the developer creates a single unit test that fails. Then, the developer creates the simplest code possible to make the test pass (without breaking the others). Once the test passes (along with all the other existing tests), the developer refactors the code to clean it up, and then repeats the cycle, starting again with a single test.

This pre-coding creation of unit tests serves as a design method. Since the tests usually exist in a separate class than the code they test, creating the test forces the developer to think about the interface his or her code provides from the perspective of using it from another class. It also encourages the developer to identify the domains of applicable values for the code, and the circumstances that should cause exceptions. Another benefit of unit tests often cited by proponents is the idea of tests as documentation for the system. Since all code can be tied back to a particular test that required the code to be added, and the tests are executed with each change to the code, these tests are kept in sync with the code. They also serve as examples of what is expected of the code – both for inputs and outputs, and for exceptional conditions.

Both of these benefits are aided by and foster a deep understanding of the program. The initial designing of the code that goes into creating the test instills the understanding in the original developer, and this understanding then gets encapsulated in the unit tests, where it is ready for future developers to quickly regain when they have to work with the code. With the tools facilitating this degree of program understanding, it follows that these tools may be useful not only for developing a system, but also for reverse engineering it once it has been developed.

3. The REEF

Space limitations on this paper preclude an in-depth description of the REEF's entire structure here; the interested reader is referred to [16] for more information. The framework classifies reverse engineering tools across five major dimensions: cognitive model support, reverse engineering tasks, canonical activities, quality attributes, and miscellaneous characteristics. This paper focus on reverse engineering tasks, canonical activities, and quality attributes only.

There have been a number of assessment instruments developed whose goals include an evaluation of the capabilities inherent in reverse engineering tools to support activities related to program understanding (e.g., [7],[8],[10]). The REEF has been used to examine several different applications domains (e.g., [15][17][18]). This paper represents the first time the REEF has been used in the context of unit testing.

Reverse Engineering Tasks

The REEF identifies five reverse engineering tasks as being the most important ones to consider. These tasks are program analysis, plan recognition, concept assignment, redocumentation, and architecture recovery. Program analysis is pattern matching at a syntactic level, plan recognition is pattern matching at a semantic level in the programming level domain, and concept assignment is pattern matching at a semantic level in the application domain. Redocumentation is the recreation of documentation for an existing system, while architecture recovery recovers the overall design (architecture) of the existing system.

Canonical Activities

The activities of reverse engineering can be organized into three main groups: data gathering, knowledge management, and information exploration . These groups are derived from the classification of the artifacts manipulated by reverse engineering into three categories. Artifacts can be data (factual information); knowledge (data along with relationships and rules derived from that data); and information (knowledge which is communicated based on a contextual selection from all available knowledge).

Quality Attributes

Practitioners often refer to the attributes as the "-ilities" as many of the attributes share that suffix. Dependability, reliability, and usability are all examples of a quality attribute. The REEF includes three of these attributes as measures on which to compare tools: applicability, extensibility, and scalability. Applicability examines the subset of application and implementation domains to which a given tool applies. Extensibility looks at the mechanisms a tool provides for its end user to extend and customize the tool's capabilities, and scalability looks at how well a tool can handle projects of increasing size and complexity.

4. Assessment

Consider the following scenario: a developer is tasked with understanding a Java program. This system may or may not have much in the way of accurate documentation, and it is the developer's task to perform some maintenance action on the system. Because schedules are tight, the developer needs to get the required understanding of the system as quickly as

possible, and ideally, there would be some method of capturing the understanding to make it easier for the next person who has to maintain the system.

The tools discussed in this section can help with this task. Although the tools are primarily for unit testing, they do have varying degrees of reverse engineering capabilities that can be used in support of program understanding. This section describes an assessment of these capabilities according to the REEF.

4.1 Agitator

Agitar Software's Agitator is a tool designed to "agitate" code in order to find bugs in the code. This agitation involves executing the code with various inputs that are automatically generated by the tool. While the code is being executed, Agitator makes observations of conditions which are always or often true of the code. These observations are then presented to the user who determines which are supposed to be always true. The observations that are supposed to be true are promoted to assertions, and all further agitation runs of the code check that the defined assertions always hold true. Assertions can be assigned within specific contexts of the application (so that a path that generates an exception could have a different set of assertions than a normal path through the code).

Reverse engineering tasks

Program Analysis. Agitator focuses on individual methods of the classes it is testing and the parameters of those methods. It also allows outcome partitioning of the assertions. An outcome partition is one possible path through the code. By default, Agitator provides outcomes for the normal path through the code and any exceptions it encounters while executing the code. Additional outcomes can be created, however, lending support to more complex situations and allowing for more fine-grained assertions to be made about the code. Both this partitioning of outcomes and the partitioning of the assertions by class and method are examples of tasks that can be categorized as a type of program analysis. In both cases, it aids in the understanding of the program by narrowing the engineer's focus to one small part of the code, and the assertions that are relevant to that one part.

Plan Recognition. Another reverse engineering task that Agitator supports is plan recognition. The rules that define where the experts are applicable serve as patterns that the tool then matches against the system. These rules allow filtering based on the names of classes or the inheritance hierarchy of the code before an applicability check is performed. This applicability check can then apply any programmable criteria to determine if the expert applies to the project, class, or method being examined. Essentially, the rules are defining the clichés to which the expert is relevant.

Redocumentation. A third reverse engineering task supported by Agitator is redocumenting the program through the creation of observations and assertions. As observations are made and assertions identified about the program under certain circumstances, these assertions can then be recorded in Agitator. While it is difficult to get the assertions out for use in other formats, Agitator does provide the benefit of rechecking these assertions on demand. This is particularly useful as changes are made to the application–if a change is made to an application that violates one of the assertions, it is found during the next agitation run.

Canonical activities

Data Gathering. Agitator supports data gathering by the static analysis of applying the experts to the code and applying rules of coding style to the code. In addition, Agitator performs dynamic analysis of the system while the code being examined is executed and the observations are being made about the code. This feature aids in the act of reverse engineering as well, by displaying the observations that it finds to be true or almost always true (along with the values for which an observation was not true).

Knowledge Management. The discovery element of knowledge management is supported through the creation of observations. As Agitator observes things about the application, it may reveal information or patterns that even the original developers did not recognize. Agitator also helps in the evolution of knowledge about the system, particularly as the system changes. With a click of the mouse, the code can be re-agitated. During this process, all the existing assertions are verified to still be true, and new observations may be made. These new observations or failing assertions may reflect updated information about the code (thus replacing old information) or they may reflect bugs that have been introduced into the application.

Information Exploration. Agitator supports information exploration by navigating through the information about the code using the considerable navigation capabilities built into Eclipse. Another element of the information exploration activity is that of analyzing the information. For reverse engineering, Agitator largely does this through the assertions and observations, though its flagging of code style violations also does some analysis as well. The final element of this activity is that of the presentation of information.

Agitator has multiple ways of presenting its information —via Eclipse views displaying different aspects of the agitation process, via graphical display within the source editor itself, indicating how much coverage was achieved by the agitation and presentation of information through its Management Dashboard (a set of generated Web pages providing overviews of the project).

Quality attributes

Applicability. Agitator only works for programs written in Java. Because it is developed, marketed and priced as a commercial software package, it is more applicable to projects that have a substantial tools budget, rather than the majority of open source or freeware software being developed. Agitar does remedy this to some extent by agitating some common open source code and making the dashboards available for these projects on their website.

Extensibility. Agitator exists as a set of Eclipse plug-ins. Because of this, it is possible to augment the functionality of the tool by installing one or more of the many existing Eclipse plug-ins [5]. Agitator also provides a high degree of end-user programmability. While it provides many built-in factories and experts, it also includes APIs so that end-users can develop their own instances of these components as their project requires them. It is also possible for additional assertions to be created, using a Java-like syntax. This allows the end user to program in assertions that are not based on observations.

Scalability. Agitator scales to large projects fairly well. Since each set of assertions applies to a single outcome partition of a single method, and agitation is done per class, it is feasible to have many classes and methods while still focusing on just one at a time.

4.2 JUnit

JUnit is an open-source tool created by Erich Gamma and Kent Beck, and is a member of a family of tools collectively referred to as "xUnit" that span a variety of programming languages. JUnit provides a set of methods the developer can use to assert certain conditions on the code. These assertions typically are based on return values of methods in the code, but may also involve other aspects of the environment or the system being tested.

Reverse engineering tasks

Program Analysis. JUnit's main function is to execute tests against the application being tested. Each time these tests are run provides an instance of the reverse engineering task of program analysis. The tests analyze the program, indicating what's working as expected and what is not. For reverse engineering, this is useful if the tests were created along with the code. It is also useful if maintenance has begun and changes are being made to the code.

Redocumentation. The agile development community argues that the code in the unit tests serves as documentation and they are examples of how to call into the methods being tested and a complete set of unit tests illustrates the boundary conditions and constraints to which the system is subject. With this idea of tests as documentation, creating unit tests for an existing system can be categorized as an element of the redocumentation reverse engineering effort. At the end of the creation process, an artifact (the test suites and cases) remain to serve as documentation for the code, reducing the time required for later engineers to work on the system.

Canonical activities

Data Gathering. The results of executing the tests also serve as a source of data for the canonical activity of data gathering. The tests serve as dynamic system examination, and give insight into what works or what hypotheses are correct.

Knowledge Management. JUnit's tests also serve the canonical activity of knowledge management. They can be used to organize knowledge that has already been obtained by grouping tests of similar functions or contexts together. The tests can also be used to discover additional knowledge about the application through the encoding of hypotheses in the tests. As the tests pass or fail, they confirm or disprove the hypotheses. Each of these results adds to the knowledge available about the system. The tests can be used to encapsulate experience about the system once it is gathered. These tests can either be pre-existent (created when the code was originally written, as in TDD) or created based on the results of other experience capture activities.

Quality attributes

Applicability. The xUnit family works with many languages and each individual tool applies in much the same way as described above. JUnit, in particular, works with Java programs. By itself, it can be used for the public and default level methods and fields of a class. However, add-ons are available that extend the tool's reach to GUIs and private members [11].

Extensibility. JUnit is released as an open source tool which means that anyone can download the source code and extend the tool as they desire. It is also possible to extend the default classes (through add-ons), leaving

the core JUnit code as is, and thus increasing the potential user base for additional functionality.

Scalability. As with Agitator, JUnit tests focus on no more than one unit each (and often there may be several tests for that unit). This means that the only issue with performance scaling is one of the time involved, both to develop the tests and to execute them. Large suites of JUnit tests may take 5-10 minutes to execute, but usually the tests run in less time than that.

4.3 EasyMock

EasyMock is an open-source tool written by Tammo Freese. It serves as more of a support tool for unit testing than a tool for executing unit tests. EasyMock allows its users to create a mock object which functions as a proxy for a real object that is unavailable, unpredictable, or otherwise difficult to test.

Reverse engineering tasks

EasyMock's lack of standalone reverse engineering capabilities means that it does not directly perform any of the REEF's reverse engineering tasks. Instead, EasyMock supports the tasks of program analysis and redocumentation by allowing the engineer to focus on one piece of the application. It also provides the ability to better control the return values of subsystems (such as databases, for example) that make it easier to understand the error handling code present in an application.

Canonical activities

Data Gathering. EasyMock aids in the canonical activity of data gathering by performing dynamic system examination. When a test containing EasyMock code is executed, EasyMock examines the method calls made to the part of the system that is being mocked. It then validates these calls against the ones it was told to expect. This can be helpful in understanding the flow of actions in the system, particularly if the tests are developed in an evolutionary fashion where few or no expected calls are described, and each call that EasyMock reports is verified and then added to the test.

Knowledge Management. EasyMock provides a limited form of knowledge management. As the tests are created with accurate expected results, they store the knowledge of the order that went into creating the tests. As more method calls are discovered, the tests can be updated to reflect evolving knowledge about the system.

Information Exploration. EasyMock can aid information exploration. Each time an EasyMock test executes, it analyzes the inter-method calls being made into the portion of the system being mocked. It then confirms or disproves that these calls occur in the order and with the parameters that are expected.

Quality attributes

Applicability. EasyMock can be used on any Java application. Its design makes it much more suited to cases where interfaces are used and classes are passed into methods, and without the use of these language features, substantial code changes may be required to use the mock object created by EasyMock. These changes run the risk of introducing bugs into the code being reverse engineered, potentially taking the reverse engineer down a path away from program understanding.

Extensibility. EasyMock is released as open source software, meaning that anyone can extend its functionality by editing the code.

Scalability. Like the other tools in this paper, EasyMock focuses on units of code. This narrow focus allows it to scale well to a large number of units, though the time required for deploying and using the tool does increase with the number of units.

5. Summary

This paper described an evaluation of the capabilities of three unit testing tools in a situation where reverse engineering of a system was required. A summary of the canonical activity portion of the evaluation across all three tools is included in Table 1.

Our evaluation of these tools revealed that unit testing tools share several characteristics that make them useful for reverse engineering in aid of program understanding. For example, they have a narrow focus, which allows them to scale up to large systems fairly easily. The tools focus on units of code, which means that the tools have the capability to gain access the individual units. Because of the emphasis of frequent execution of the tests, unit testing tools often serve to encapsulate knowledge about the system that would otherwise be lost. This aids in future maintenance efforts by reducing the time it takes to get back up to speed.

Some areas that may prove useful to examine in the future is that of the variation in language focus amongst unit testing tools. For example, while the majority of the xUnit family of tools are modeled on JUnit, some tools are now beginning to evolve in different directions based on the capabilities of their base languages. Nunit [4], and TestNG [13] are trying to improve on the model presented by JUnit and may have different impacts on reverse engineering.

Finally, an area that could prove worthy of study is the opposite of this research: the application of reverse engineering tools to software testing. Since testers are gaining understanding of what a program does when they test it, it is quite possible that some of the tools of reverse engineering might prove very helpful to them.

References

[1] EasyMock, online at http://www.easymock.org

[2] Jemmy, online at http://jemmy.netbeans.org/

[3] JUnit, online at http://www.junit.org

[4] NUnit, online at http://www.nunit.org/

[5] "Eclipse Plugins."

[6] Agitator, online at http://www.agitar.com

[7] Armstrong, M. and Trudeau, C. "Evaluating Architectural Extractors." *Proceedings of the 5th Working Conference on Reverse Engineering* (WCRE '98).

[8] Bellay, B. and Gall, H. "A Comparison of Four Reverse Engineering Tools." *Proceedings of the 4th Working Conference on Reverse Engineering* (WCRE '97).

[9] Extreme Programming. URL: http://xprogramming.com/

[10] Gannod, G. and Cheng, B. "A Framework for Classifying and Comparing Software Reverse Engineering and Design Recovery Techniques." *Proceedings of the 6th Working Conference on Reverse Engineering* (WCRE '99).

[11] JUnit-Addons, online at http://junit-addons.sourceforge.net/

[12] Scrum, online at http://www.controlchaos.com/

[13] TestNG, online at http://testng.org/

[14] Tilley, S. "A Reverse-Engineering Environment Framework". Carnegie Mellon University CMU/SEI-98-TR-005, April 1998.

[15] Tilley, S. "Discovering DISCOVER." Carnegie Mellon University CMU/SEI-97-TR-012, October 1997.

[16] Tilley, S. "The Canonical Activities of Reverse Engineering". *Annals of Software Engineering*, vol. 9, pp. 249-271, 2000.

[17] Tilley, S. and DeSouza, M. "Spreading Knowledge about Gnutella: A Case Study in Understanding Net-Centric Applications." *Proceedings of the 9th International Workshop on Program Comprehension* (IWPC 2001: May 12-13, 2001; Toronto, Canada), pp. 189-198. Los Alamitos, CA: IEEE Computer Society Press, 2001.

[18] Tilley, S. and Huang, S. "Evaluating the Reverse Engineering Capabilities of Web Tools for Understanding Site Content and Structure: A Case Study." *Proceedings of the 23rd International Conference on Software Engineering* (ICSE 2001: May 12-19, 2001; Toronto, Canada), pp. 514-523. Los Alamitos, CA: IEEE Computer Society Press, 2001.

			Agitator	JUnit	EasyMock
Data Gathering	System Examination	Static	Code rules, expert application	No	No
		Dynamic	Observation generation	Test results	Method calls
		Mixed	No	No	No
	Document Scanning		No	No	No
	Experience Capture		Observations/Assertions	Tests	Expected results
Knowledge Management	Organization		Hierarchy of assertions	Tests	No
	Discovery		Observations	Tests	Actual call order
	Evolution		Rerunning assertions	Changes detected	Changes detected
Information Exploration	Navigation	Selection	Eclipse features	No	No
		Editing	Eclipse features	No	No
		Traversal	Eclipse features	No	No
	Analysis	Types	Assertions, observations, code rules	Functionality	Inter-method communication
		Levels	Yes	No	No
		Automation	Yes	Yes	Yes
	Presentation	Multiple Views	Yes	With add-ons	No
		Visualization Techniques	Several views	Green bar/Red bar	JUnit results
		User Interface	Partially changeable	Fixed	Fixed

Table 1: Canonical Activities

Estimating Event Lifetimes for Distributed Runtime Verification

Christos Kloukinas, George Spanoudakis, Khaled Mahbub
Department of Computing, The City University, London, EC1V 0HB, UK
{C.Kloukinas, G.Spanoudakis, K.Mahbub}@soi.city.ac.uk

Abstract

Runtime system verification has been proposed as a form of dynamic verification of software systems which can be applied in settings where complete static verification or exhaustive system testing is not practical. Runtime verification checks properties against runtime events generated during the operation of a system. Current approaches to runtime verification assume that runtime events are time-stamped by a single clock and, thus, can be totally ordered. They also assume that events are received by the reasoning engine in the same order as they have been produced. These assumptions are apparently true only in systems with a single clock. In this paper, we present the extension of a framework for runtime verification which can monitor distributed systems, in which events are produced by different components, each having its own clock.

1. INTRODUCTION

Runtime (or dynamic) system verification has been proposed as a complementary approach to static system verification and testing, which can enhance confidence in the correctness of system operations by monitoring and identifying violations of required system properties during the normal system operation [3][6][10]. Runtime verification is needed due to the inability to guarantee the completeness of system models that have been used for static analysis and the preservation of these models by system implementations. It is also useful as it is difficult to foresee all the different circumstances that may arise during the operation of a system and therefore guarantee that the assumptions, under which its correctness can be statically proved, hold at runtime.

Typically, platforms for runtime verification (e.g. [6] [9][11][20]) provide support for specifying formally the properties of a system that should be verified at runtime, identifying the events that should be available in order to assess if certain properties are satisfied, capturing these events at runtime, and checking for violations of the required properties.

The main limitation of existing runtime verification platforms is that they assume that the systems to be monitored consist of components running on a single machine. In such cases, the events of the system that is being monitored are: (i) time stamped by a single clock, (ii) totally ordered, and (iii) received by the monitor in the same order as they are generated by the system that is being monitored.

Whilst valid in the case of centralised systems, these assumptions do not necessarily hold in cases of distributed systems with components running on different platforms. In such systems, runtime events may come from distributed components operating with different time clocks. Furthermore, distributed system components may have different types of connections with the monitor and, therefore, generate events which arrive at the monitor with different communication delays and possibly in an order that is not the same as the order of their generation.

Thus, in order to check properties involving events from distributed components, a monitor would have to overcome two problems: (i) to synchronise the clocks of the various event sources, so that the timestamps of the different events can be ordered and compared to each other, and (ii) to establish until when a particular event needs to be stored, so that it can reason about the system properties in a sound way or, equivalently, to compute the required monitoring lifetime of each event.

Consider, for instance, the case of monitoring the availability of the communication channel between two components C1 and C2 of a system by ensuring that the dispatch of a request R from C1 (*Event-1*) will always be followed by the receipt of R by C2 (*Event-2*) within a specific time period. In this case, *Event-2* may arrive at the monitor before *Event-1* due to different communication delays in the relevant channels. Thus, when the monitor receives *Event-2* it will have to decide for how long it should wait for *Event-1* and wait for this event before dropping *Event-2* or otherwise it may report a false violation of the availability of the communication channel between C1 and C2. This would happen in cases where, after dropping *Event-2*, the monitor receives an *Event-1* corresponding to it.

In this paper, we present an extension of a dynamic verification framework described in [20] which addresses these problems. The original framework monitors systems against properties expressed in Event Calculus (EC) [19] and was initially developed to support monitoring based on events which are generated by a single source. The extension of the framework that we present in this paper enables it to monitor systems in which events are generated by distributed sources having different clocks and communication channels to the monitor.

The rest of the paper is organised as follows. In Section 2, we provide an overview of our monitoring framework and the language that it uses to specify monitoring properties. In Section 3, we propose a solution for

computing the lifetime of events that the framework receives from distributed sources and show how these lifetimes are used during monitoring. In Section 4, we give an overview of related work and, finally, in Section 5 we summarise our work and outline directions for further research.

2. MONITORING FRAMEWORK

2.1. Overview

As shown in Figure 1, the dynamic verification framework that we have extended consists of a *monitoring manager,* a *monitor,* a *Network Time Protocol (NTP) server*, and communicates with different event collectors attached to the components of the system that is being monitored.

The *monitoring manager* has responsibility for initiating, coordinating and reporting the results of the monitoring process. Once it receives a request for monitoring a specific set of properties, the manager checks whether it is possible to monitor them and, if it is, it sends the properties to be checked to the monitor, and starts listening to events which are generated by different types of external event collectors. These events are received via TCP/IP sockets and sent to the monitor.

After receiving events from the manager, the *monitor* checks whether they violate any of the properties given to it. The monitor is a generic engine for checking violations of EC formulas against a given set of runtime events. During monitoring, it also takes into account information about the state of a system that it derives from runtime events using a special type of EC formulas called *assumptions* (see Section 2.2). When a violation of a property is detected, the monitor records it in a *deviation database* which is polled regularly by the monitoring manager to retrieve detected deviations.

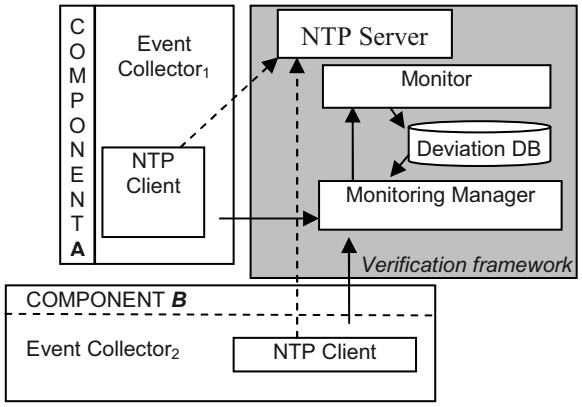


Figure 1 : Verification framework

The framework assumes that the components of the systems to be monitored have associated *event collectors* that can capture events during their operation and send them to the monitor. When a collector captures a runtime event, it wraps it into an envelope with additional information including the source of the event (i.e., the component where it was captured) and a timestamp indicating when the event was captured at the component.

To enable the synchronisation of event timestamps, the framework incorporates components that realise the *Network Time Protocol* [17] (i.e., a protocol based on the clock synchronisation scheme described in [12]). The implementation of this protocol allows event collectors to compute the difference of their clocks with the clock of the monitor at regular intervals. This difference is used to transform timestamps taken according to the clock of each collector into timestamps that express time in terms of the monitor's clock. This is achieved by implementing an *NTP client* at each event collector and an *NTP server* at the machine that hosts the monitor, as shown in Figure 1. The NTP clients call the NTP server at regular intervals to synchronise their clocks with the clock of the server. The use of NTP can synchronise distributed clocks at a very high level of accuracy since recent versions of NTP (version 4) use a resolution of less than one nanosecond.

2.2 Specification of Properties

As indicated in Section 1, in our runtime monitoring framework the properties to be monitored are expressed in a language based on Event Calculus (EC) [19]. EC is a first-order temporal logic language which can be used for representing and reasoning about *events* and their effects over time. An event in EC is an occurrence that takes place at a specific instance of time (e.g., invocation of a system operation, receipt or dispatch of a message) and may have an effect. The effects of events are represented by *fluents*. Fluents are conditions which may change over time (e.g. a condition indicating that a system has received a message) and are initiated and/or terminated by events.

The occurrence of an event in EC is represented by the predicate *Happens(e,t,ℜ(lb,ub))*. This predicate denotes that an instantaneous event *e* occurs at some time *t* within the time range *ℜ(lb,ub)* (i.e., lb ≤ t ≤ ub) . The boundaries *lb* and *ub* that define time ranges are specified as linear expressions over time variables of *Happens* predicates in an EC formula of the form:

$$lb = l_0 + l_1 t_1 + l_2 t_2 + \ldots + l_n t_n$$
$$ub = u_0 + u_1 t_1 + u_2 t_2 + \ldots + u_n t_n$$

Given our focus on runtime system monitoring, the events we consider represent invocations of system operations, responses from such operations, or exchanges of messages between different system components. Thus, events have the following structure which captures the information required for monitoring such system interactions without affecting the overall expressiveness of the framework with respect to standard EC:

event(_id, _sender, _receiver, _status, _sig, _source)

In this structure:

- *_id* is a unique identifier of the event
- *_sender* is the identifier of the system component that sends the message represented by the event
- *_receiver* is the identifier of the system component that receives the message represented by the event
- *_status* is the processing status of an event (i.e. whether or not its processing has started when the monitor receives it)
- *_sig* is the signature of the dispatched message or the operation invocation/response represented by the event, comprising the operation name and its arguments/result.
- *_source* is the identifier of the component where the event was captured.

Fluents are defined as relations between objects of the form $rel(O_1, ..., O_n)$ where *rel* is the name of a relation which associates the *n* objects O_1, …, and O_n. The initiation or termination of a fluent *f* due to the occurrence of an event *e* at time *t* is denoted in EC by the predicates *Initiates(e,f,t)* and *Terminates(e,f,t)*, respectively. An EC formula may also use the predicates *Initially(f)* and *HoldsAt(f,t)* to denote that a fluent *f* holds at the start of the execution of a system and at time *t,* respectively.

The rules to be monitored at runtime are specified in terms of the above predicates and have the general form $body \Rightarrow head$. The meaning of a rule is that if its body evaluates to true, its head must also evaluate to true. The *Happens* predicates in a rule which have no constraints for their lower and upper time boundaries are what we call "unconstrained" predicates. During the monitoring process, rules are activated by events that can be unified with the unconstrained *Happens* predicates in them. When this unification is possible, the monitor generates a rule instance to represent the partially unified rule and keeps this instance active until all the other predicates in it have been successfully unified with events and fluents of appropriate types or it is deduced that no further unifications are possible. In the latter case, the rule instance is deleted. When a rule instance is fully unified, the monitor checks if the particular instantiation that it expresses is satisfied.

An example of a rule that can be expressed in the EC language of our framework is given by the formula below:

Rule 1: ∀ _eID1,_C1,_C3:String; t1:Time
Happens(e(_eID1,_C3,_C1, REQ, authorise(),
_C3),t1,ℜ(t1,t1)) ⇒ ∃ _eID2:String ; t2:Time
Happens(e(_eID2,_C3,_C1, REQ, authorise(),
_C1),t2,ℜ(t1+1,t1+10))

This rule states that when an event *e(_eID1,_C3,_C1, REQ, authorise(), _C3),t1,ℜ(t1,t1))* representing a call of the operation *authorise()* in a component *_C1* by a component *_C3* is dispatched, it must be followed by an event *e(_eID2, _C3, _C1, RES, authorise(), _C1)* represening the receipt of the call by *_C1* in no more than 10 time units after the dispatch of call. Thus, *Rule 1* expresses a bounded availability property for the communication channel between the component *_C3* and other components of the system (*_C1*) since it requires that the requests generated by *_C3* are transmitted within a bounded time period.

The unconstrained predicate in this rule is the predicate $Happens(e_1^{C3}, t1, \Re(t1,t1))$[1], since the lower and upper bounds of its time variable are defined without any references to other time variables in the rule. Thus, at runtime, new instances of *Rule 1* will be generated as soon as an event that can be unified with this predicate is received. Each of these rule instances will remain alive until it is fully unified or until no further unification of an event representing the receipt of a response of the call dispatched by *_C3* in the rule instance is possible.

Note that, as in the above example, our framework requires all the constrained predicates in a rule to have time variables with constrained upper bounds. This is to ensure that rules can be verified. For example, if the *Happens* predicate for e_2^{C1} in the head of *Rule 1* did not have an upper bound, then its absence would never cause the monitor to flag the rule as violated, since the monitor would always wait for some e_2^{C1} event at some point in the future.

3. COMPUTING LIFETIME OF EVENTS

As we discussed in Section 1, the problem that arises with the use of events which are generated by distributed sources is two-fold: firstly we need to synchronise the clocks of the different event sources so that the timestamps of the events that they generate can be comparable to each other and secondly we need to know until when we need to store a particular event in order to be able to reason about the system state and check rules. The clock synchronisation that is performed through the use of the Network Time Protocol (NTP) by our framework solves the first problem but not the second.

To appreciate the second problem, consider *Rule 1*, assuming without loss of generality that *_C3* and *_C1* denote both the source of the event and the clock of the source system component where the event was captured. As the occurrence of events of type e_1^{C3} in *Rule 1* is unconstrained, events of this type can instantiate the rule during monitoring. Unlike them, events of type e_2^{C1} are temporally constrained by e_1^{C3} events in the rule and cannot, therefore, create new instances of the rule; they can only be unified with existing rule instances.

[1] e_1^{C3} is an abbreviated reference to the event *e(_eID1,_C3,_C1, REQ, authorise(), _C3)*, in which the subscript denotes to the event ID and the superscript to the event source. Such abbreviated references are used in the rest of the paper in all cases where other event variables are not important.

Thus, if the monitor receives an event of type e_2^{C1}, in addition to unifying it with all the current instances of *Rule 1*, it must keep it until there is no possibility to receive an e_1^{C3} event that could be correlated with e_2^{C1} through *Rule 1*. This is necessary since if e_2^{C1} is dropped and later the monitor receives an e_1^{C3} event with an earlier timestamp than e_2^{C1}, it would report a false violation of *Rule 1*. The possibility of e_1^{C3} and e_2^{C1} events arriving at the monitor in the opposite order of their occurrence arises due to different (and dynamically changing) communication delays in the channels that connect C_1 and C_3 with the monitor or even attacks in these channels that can cause the loss of events. Whilst keeping events of type e_2^{C1} in this case is necessary for the soundness of the monitoring results, the monitor must also ensure that it keeps these events only for the maximum time that is necessary for the soundness of the results. This is because if the monitor keeps them longer the size of its event store will increase monotonically with a deteriorating effect on both the space and time required for monitoring. The maximum time point until when events e_2^{C1} would need to be kept by the monitor, in this example, can be established by finding the maximum value of the time variable t1 of e_1^{C3} events that satisfies the constraints: (1) $t1 \leq t2-1$ and (2) $t2 \leq t1+10$.

In general, for a rule with *n+1 Happens* predicates, there will be *2n+1* such inequalities to solve. This is because one of the rule predicates is unconstrained (the one firing the rule), the remaining *Happens* predicates contribute two inequalities each, and we need an extra equality to establish the exact value for the time variable of the event in question (t2 in our previous example with the e_2^{C1} events).

Figure 2 shows the algorithm for computing the lifetimes of an event. According to this algorithm, when an event *e* occurs, the set *R(e)* of rules which have predicates that can be unified with the event *e* is determined (this set includes rules that have event types which are the same as the type of *e* or supertypes of it). The set *R(e)* will include rules that may specify time constraints for the event that cannot be fully evaluated yet. Subsequently, the constraints of each rule in *R(e)* are identified and expanded with an equality expressing that the time variable of the predicate of the rule that has been unified with *e* is equal to the timestamp of *e* (step 2.a). Given the time constraint set that results from this process, the algorithm computes the maximum possible value for each of the time variables of the rule using the *Simplex* method [8] (step 2.b.i). Subsequently, it groups the different time variables according to the clock of the event source they are related to (step 3), and generates a set of all the conditions (*Lifetime(e)*) for computing the upper bound of the lifetime of *e* (step 4). A condition in *Lifetime(e)* states that *e* won't be needed after the last event that is seen from a channel which is relevant to *e* has a timestamp ($last\_observed(c_j)$) that is greater than the maximum possible value of the time variables grouped in this channel's group (see condition $last\_observed(c_j) > max_{t_i \in G_j}(max(t_i))$). The reason for using the timestamp of the last event that has been observed from a clock in the evaluation of the *Lifetime(e)* conditions is because events are communicated to the monitor through TCP/IP sockets which *guarantee* a FIFO transmission within the same component (clock)-monitor channel. The conditions in *Lifetime(e)* determine the lifetime of e since when their conjuction becomes true, the lifetime of *e* will expire.

Compute_Lifetime(e):

1. $R(e) = \{ r \mid r$ has a predicate p that can be unified with $e\}$
2. Forall $r \in R(e)$ do
   a. $CN_r = \{$time constraints of $r\} \cup \{$time variable of predicate p that matches e = timestamp of $e\}$
   b. Forall $t_i \in CN_r$ do
      i. Find $max(t_i)$ given CN_r
3. Group the time variables t_i into as many groups G_j as the different event sources (clocks) c_j in $R(e)$
4. $Lifetime(e) = \cup_j ((last\_observed(c_j) > max_{t_i \in G_j}(max(t_i))))$

Figure 2: Computing the lifetime of an event – I

In our previous example, if *Rule 1* is the only rule that is being monitored and an event of type e_2^{C1} is observed at t2=10, step 1 will produce the set $R(e_2^{C1}) = \{Rule\text{-}1\}$, step 2.a will produce $CN_r = \{t1 \leq t2-1, t2 \leq t1 + 10, t2 = 10\}$, step 2.b.i will produce the solutions $max(t1)=9$ and $max(t2)=10$ by finding the maximum value of t1 for which the constraints in CN_r are satisfied, and step 3 will produce two groups of time variables $\{t1\}$ and $\{t2\}$, for the two clocks C_1 and C_3, respectively. Finally, in step 4, the lifetime constraint set for e_2^{C1} will be established as:
$Lifetime(e_2^{C1}) = \{(last\_observed(C1)>10), (last\_observed(C3) > 9)\}$.

It should be noted that our algorithm uses the Simplex method, which has exponential complexity $O(2^n)$ (for a problem with *n* variables [8]), to find the maximum time of a time variable in *step 2.b.i.*, although there are algorithms with polynomial compexity (the worst case complexity of Karmarkar's algorithm [1], for example, is $O(n^{3.5})$). This is because for small numbers of variables, as the ones normally appearing in monitoring rules ($n \leq 10$), Simplex has better performance. Furthermore, the algorithm of Figure 2 computes the maximum value of a time variable for each rule separately, rather than combining them into a single larger problem. This is because the individual rule problems can be solved independently and a larger set of rules would take more time to solve due to the additional time variables (in general $2^n + 2^m < 2^{n+m}$ for $n,m \geq 2$). Due to this approach, once the individual rule systems have been solved, the different time variables that are associated with events

coming from the same clock need to be grouped together, as done in step 3 of the algorithm.

Note that at this step, the algorithm of Figure 2 assumes that the clocks/sources of the events in the rules are fully specified when a rule is matched with an incoming event. In the example of *Rule 1* this is the case, since the sender of an event e_2^{C1} (i.e. C3) is also the source of events e_1^{C3}. Thus, when *Rule 1* is matched with an e_2^{C1} event, the identity of C3 becomes known. However, there might be cases where the exact source of events that could potentially be matched with a rule is not known after the rule is matched with arrived events. Consider, for instance, the following rule:

Rule 2:

∀ _eID1, _eID2, _U: String; _C1,
_C3: Terminal; _C2: Component; t1, t2:Time
Happens(e(_eID1,_C1,_C2, REQ, login(_U,_C1),
_C1),t1,ℜ(t1,t1)) ∧
Happens(e(_eID2,_C3,_C2, REQ, login(_U,_C3),
_C3),t2,ℜ(t1,t2)) ∧ _C1 ≠_C3 ⇒ ∃ _eID3: String; t3:Time
Happens(e(_eID3,_C1,_C2, REQ-A, logout(_X,_C1),
_C1),t3,ℜ(t1+1,t2-1))

This rule requires that if a user *_U* logs in to a system *_C2* from a terminal *_C1* and later he/she logs in again from a different terminal *_C3*, he/she must have logged out from the former terminal before the second login. The rule effectively monitors cases where users are logged in from different terminals at the same time. When an event *e(_eID2,_C3,_C2, REQ, login(_U,_C3), _C3)* (or e_2^{C3} in our abbreviated form) arrives at the monitor, its lifetime will need to be estimated in reference to the maximum possible values of time variables *t1* and *t3*. In this case, however, the algorithm of Figure 2 does not work, since at step 3 it is not known which other terminals the user of e_2^{C3} may be using or, equivalently, which source clocks should be associated with the time variables *t1* and *t3*.

Compute_Lifetime(e):

1. *R(e) = { r | r has a predicate p which unifies with e}*
2. Forall $r \in R(e)$ do
   a. CN_r= {time constraints of r} ∪ { time variable of predicate p that matches e = timestamp of e}
   b. Forall $t_i \in CN_r$ do
      i. Find $max(t_i)$ given CN_r
3. Group the time variables t_i into as many group types TG_u as the different types of event sources c_u in *R(e)*
4. Forall group types $g \in TG_u$ do
   c. Forall the known sources *j* of type *g* do
      i. Create a group G_j and assign copies of the time variables of *g* to it
5. $Lifetime(e) = \cup_j \{(last\_observed(c_j) > max_{t_i \in G_j}(max(t_i)))\}$

Figure 3: Computing the lifetime of an event – II

To deal with such cases, we use an extension of the algorithm, shown in Figure 3. The extended algorithm initially groups time variables into groups corresponding to the types of the event sources that are associated with them in the rules. Then, for each of the source type groups, it finds all the sources of the particular type that are known to the system, creates different groups for them and assigns copies of the time variables of each source type to each of the source groups that were generated from the type. Thus, if it is known that the system that is being monitored with *Rule 2* has 3 terminals, the algorithm of Figure 3 will create different variable groups for each of these terminals and assign copies of the time variables *t1* and *t2* to each of these groups.

Having computed the *Lifetime(e)* constraint set, upon the arrival of an event *e* at runtime we use it to compute a vector with the maximum time values for *e* with respect to the different clocks related to it. For the ongoing example of *Rule 1*, the vector of e_2^{C1} would be < 10, 9 >. The event and its vector are then stored in the database of the monitor. Also, when a new event arrives at it, the monitor checks if the lifetime of some other events which depend on the clock of the new event has expired. The above process is shown in Figure 4.

1. Observe an event *e*
2. Update the global vector of observed clock values
3. Lifetime(*e*) = Compute_Lifetime(*e*)
4. Store *e* in the DB with its vector of different clock limits
5. Remove events from the DB if their clock limits have been exceeded

Figure 4: Using event lifetimes

4. RELATED WORK

Forms of dynamic verification have been developed and investigated in the context of program verification, safety critical, and service centric systems.

In program verification, research has focused on the development of programming platforms with generic monitoring capabilities, including support for generating program events at runtime (e.g. jMonitor [11]), embedding specifications of monitorable properties into programs and producing code that can verify these properties during program execution (e.g. monitoring-oriented programming [4] and [6]). There is also work focusing on runtime verification of requirements specifications [7]. However, metric time is not considered in [4], [6], [7] or [11]. Runtime monitoring methods have also been applied to autonomous safety critical systems [16], as the testing of such systems is difficult and resource consuming. In service-centric systems, dynamic verification has focused on monitoring *service level agreements* (SLAs) [2][20]. In safety critical systems, early monitoring methods focused on detecting timing failures and guaranteeing system responsiveness [9][15]. Though [15] supports timing constraints, it does not support distributed monitoring. The distributed

monitoring of [9] on the other hand does not support fluents or general expressions for time and does not clarify how the bound of the size of the event histories is decided. Event correlation has also been considered in [18] where event observers are produced as transducer automata recognizing and rewriting the input events. Compared to our framework, the approach in [18] does not support fluents or metric time.

The extension of the framework in [20] with the capabilities described in this paper makes it possible to verify complex properties, based on events captured from distributed sources, thus, exceeding the capabilities of other approaches.

5. CONCLUSIONS

In this paper we have presented extensions of the monitoring framework described in [20] that render it applicable to multi-clock distributed systems. Our extensions address two of the problems of distributed systems monitoring: (1) the need for synchronizing the clocks of different event sources so that the events they emit can be correlated, and (2) the estimation of the lifetime of events within the monitor in order to ensure that unknown transmission delays of other events that may need to be correlated with them will not affect the monitoring process. To address the first of these problems, we have incorporated an implementation of the NTP protocol in our framework. To address the second problem, we compute the maximum lifetime of an event by identifying, the constraints between the time variable of the event and time variables of other events that co-exist with it in rules and solving these constraints to find the maximum possible lifetime for the event using the Simplex method (see [12] for full details).

One possible optimisation of our solution is to statically solve all the linear constraint systems at initialisation, so as to only need to instantiate the specific values of the different timestamps associated with a new event and its related rules when the event arrives, instead of solving the corresponding linear system each time. This would require a symbolic solution of the linear constraints system instead of the more straightforward numerical solution which we currently employ. For this reason we have decided against the symbolic solution in the current implementation, and intend to examine this option once we have gained more experience with the behaviour of the current implementation in a distributed setting.

6. ACKNOWLEDGEMENTS

This work has been funded by the European integrated research project Serenity (FP6-IST-2006-27587).

7. REFERENCES

[1] Adler I, et al. "An Implementation of Karmarkar's Algorithm for Linear Programming". *Mathematical Programming*, 44: 297–335, 1989.

[2] Ghezzi C., Guinea S., Runtime Monitoring in Service Oriented Architectures, In Test and Analysis of Web Services, (eds) Baresi L. & di Nitto E., Springer, 237-264, 2007.

[3] Barringer, H., Goldberg, A., Havelund, K., Sen, K. "Rule-Based Runtime Verification", 5th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'04), 2004.

[4] Chen, F., Rosu, G. "Towards Monitoring-Oriented Programming: A Paradigm Combining Specification and Implementation". In Electronic Notes in Theoretical Computer Science, 89(2), 2003.

[5] Clarke, E.M., Grumberg, O., Peled, D. "Model Checking". MIT Press 1999

[6] D'Amorim, M., Havelund, K. "Event-based runtime verification of Java programs", Proc. of 3rd Int. Workshop on Dynamic Analysis, 2005.

[7] Feather M. S., Fickas S., Van Lamsweerde A., Ponsard C. "Reconciling System Requirements and Runtime Behaviour", Proc. of the 9th Int. Workshop on Software Specification & Design, 1998.

[8] Gale D. "Linear programming and the simplex method". Notices of the AMS, 54(3):364–369, Mar. 2007.

[9] Jahanian, F., Rajkumar , R., Raju, S. C. V. "Runtime Monitoring of Timing Constraints in Distributed Real-Time Systems". Real-Time Systems 7(3):247-273, Nov. 1994

[10] Havelund, K., Roşu, G. "An Overview of the Runtime Verification Tool Java PathExplorer", In Formal Methods in System Design, 24(2):189-215, 2004.

[11] Karaoman, M., Freeman J. "jMonitor: Java runtime event specification and monitoring library". Proc. of 4th Workshop on Run-time Verification, 2004.

[12] Mahbub K., Spanoudakis G., Kloukinas C., "V2 of dynamic validation prototype". Deliverable A4.D3.3, SERENITY Project, http://www.serenity-forum.org.

[13] Marzullo K., Owicki S. "Maintaining the time in a distributed system". ACM SIGOPS Operating Systems Review, 19(3):44–54, July 1985.

[14] Mills D. L. "Network time protocol (version 3)". RFC 1305c, Network Working Group, Internet Engineering Task Force (IETF), 1992. http://www.ietf.org/rfc/rfc1305.txt?number=1305.

[15] Mok, A. K., Liu, G. "Efficient run-time monitoring of timing constraints". *In Real-Time Technology and Applications Symposium,* 1997

[16] Nelson, S., Pecheur, C. "V&V for advanced systems at NASA", TASK NO: 10 TA-5.3.3 (WBS 1.4.4.5.3), prepared for Northrop Grumman Corp,2002

[17] NTP, www.ntp.org

[18] Sanchez, C., Sankaranarayanan, S., Sipma, H., Zhang, T., Dill, D., Manna, Z. "Event Correlation: Language and Semantics", Proc. of Embedded Software (EMSOFT), LNCS 2855: 323-339, Oct. 2003

[19] Shanahan M. P. "The event calculus explained". In Artificial Intelligence Today, Lecture Notes in Artificial Intelligence, 1600:409–430, 1999.

[20] Spanoudakis G., Mahbub K.. "Non intrusive monitoring of service based systems". International Journal of Cooperative Information Systems, 15(3):325–358, 2006.

ONTOLOGY-LEARNING SUPPORTED SEMANTIC SEARCH USING COOPERATIVE AGENTS

Cheng Zhong[1] *Zilan (Nancy) Yang*[1] *Mohsen Afsharchi*[2] *Behrouz H. Far*[1]

[1] Department of Electrical and Computer Engineering, Schulich School of Engineering, University of Calgary, Canada
{czhong, zyan, far}@ucalgary.ca

[2]Department of Electrical and Computer Engineering, University of Zanjan, Iran
afsharchim@iasbs.ac.ir

ABSTRACT

In this paper we present (1) a method for semantic search supported by ontological concept learning; and (2) a prototype multiagent system that can handle semantic search and encapsulate the complexity of such process from the users. Agents which conduct semantic search on behalf of a user, deploy ontologies to organize structured and unstructured documents in their corresponding repositories. The ontology for each repository is individualized and commitment to a common ontology is not required. The agents can improve their search capability by learning new concepts from each other. This method thus allows agents dynamically establish common grounds on concepts known only to some of them. The concept learning is realized by analyzing positive and negative examples from other agents, and/or taking votes in case of conflicts in the received knowledge by involving other agents again.

Index Terms — multi-agent system, semantic search, ontology, concept learning, semantic interoperability.

1. INTRODUCTION

In contrast with the traditional keyword search technology which purely depends on the occurrence of words in documents, semantic search denotes one or more concepts in the context of other concepts. Understanding the denotation of concepts can help retrieval part of search engine understand the context of search, the activity the users is trying to perform, thus drive expectations on the categories of documents [6]. The essence of semantic search is semantic interoperation towards denotation part in the search phrase. Nowadays, general denotation procedures are realized depending on ontology-oriented means, and ontologies adopted are usually evolved and maintained in a distributed way. Thus, multiplicity of ontologies raises the issue of integration and renders the communication between peers involved in a semantic search ineffective.

Establishment of a common ontology for a certain domain is one of the cornerstone among cooperative agents (peers) participating in semantic search. However, agreeing on a common ontology may not be realistic. In multi-agent systems (MAS) research concerning agents' communication, having a common ontology is only possible when the design rationale, the concepts and meanings assigned to the concepts as well as the context of applying the concepts are shared. In other words, the agents must be designed in such a way that all the domain concepts and their meaning (i.e. semantics) should be provided in advance. In heterogeneous MAS, for a single domain, usually there is no agreement on the ontology among developers, and for several domains, the potential ontologies are large, unwieldy and may lead to less resolution and higher abstraction.

Recently, the idea of having agents *learn* concepts from each other has been suggested as a solution to improve agent communication. For example, the work in [8] suggests a method for learning a language and the work in [9] has focused on interactions between two agents to learn a single concept. In our previous work, we have presented a method for agents to learn concepts from several peer agents [1-2] and a method for verification of the learnt concepts [5].

Euzenat in [4] defines semantic interoperability as the faculty of interpreting the annotations at the semantic level, i.e. to ascribe each imported piece of knowledge to the correct interpretation or set of models. Possible levels of interoperability needed to be considered when trying to understand an expression from other systems are in ascending order of semantic intensity:

- *encoding*: being able to segment the representation in characters;
- *lexical*: being able to segment the representation in words (or symbols);
- *syntactic*: being able to structure the representation in structured sentences (or formulas or assertions);
- *semantic*: being able to construct the propositional meaning of the representation;

- *semiotic*: being able to construct the pragmatic meaning of the representation (or its meaning in context).

This model resembles humans' natural communication style that each semantic level can be achieved only if the lower ones have been traversed. For example, people can exchange useful information only if they have chosen a language, and clarified meanings of concepts which are critical to the topic. The idea of layered semantic interoperability has already been applied to the WWW [3] and is directly applied in our architecture of layered semantic search.

In this paper we present a method and a system that uses the ontology learning in a multi-layer semantic search. The architecture and learning supported search mechanism will be explained through the prototype system in Sections 2 and 3. The implementation details are provided in Section 4 followed by an example in Section 5 and conclusions in Section 6.

2. LAYERED SEMANTIC SEARCH ARCHITECTURE

Based on the semantic interoperability model [4], we have devised the layered semantic search architecture as illustrated in Figure 1. In this model peers can communicate and conduct search at various levels. This layered architecture will reduce complexity by breaking complex semantic interoperability into smaller problems; it standardizes interfaces between adjacent layers; it facilitates modular engineering and development of search tools; and

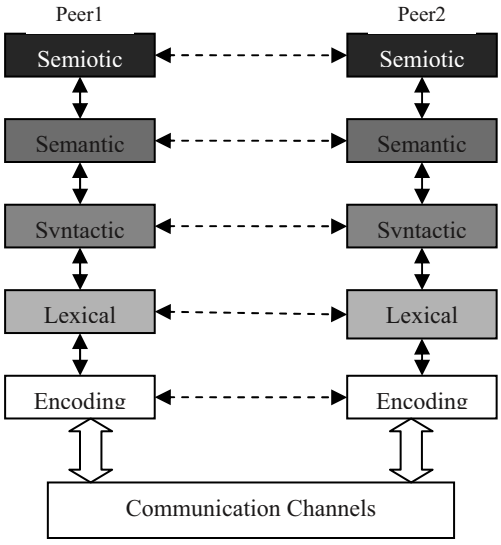


Figure 1. Architecture of layered semantic search

it accelerates evolution of technology.

The model puts some constraints on the communication between peers, namely:

1. One layer only talks with its peer layer on remote side under some agreements (or protocols). These agreements help both sides to settle natural languages, encoding standards for exchanging information, representation grammar of search phrase, etc.
2. A search phrase can be optionally initiated at any layer, then will be passed down, layer by layer, to the bottom layer (encoding layer). Each layer will add corresponding annotation information to the search phrase. Packaged phrase, finally, will be sent out.
3. Each layer can work relying on the ontology located on the same layer of semantics.

Definitions of functionalities of layers of semantic interoperability are given below.

The **encoding layer**, as base layer, defines encoding format of data exchange, thus implicitly defines the character sets of a natural language for exchanging a search phrase. ASCII and Unicode are mainly used as encoding formats. The **lexical layer** tokenizes the search phrase. At this layer, important identifiers of ontology components are identified. Functionality of this layer is not easily realizable for some natural languages such as Chinese because tokenization of sentences is a big issue due to the lack of explicit delimiter, except for punctuation, to separate each single word (or symbol). The **syntactical layer** identifies concepts by structuring words following grammar at query side, and it is capable of understanding the structured representation to extract concepts at responding side. The **semantic layer** provides ability to understand propositional meaning of the representation of search phrase. The **semiotic layer** provides ability to understand meaning of the representation of search phrase in a context (specific domain). The objective of this paper is to design and implement a prototype semantic search system to study annotation-learning workflow with focus on the lexical layer.

3. SYSTEM ANALYSIS AND DESIGN

The overview of prototype system for annotation-learning workflow within lexical layer is shown in Figure 2. In this system, software agents form a cooperative group to learn and search concepts. Each agent is responsible for a local unstructured data repository with a local ontology. The agents are responsible for organizing documents in their own repositories using any ontology they deem appropriate. The agents also communicate with each other to respond to search queries.

The system design assumptions, following the FIPA guidelines (http://www.fipa.org) are:

a) MAS is a close cooperative group. It means that there is, at least, one agent taking charge of registering service. Any other agent joining the group needs to register by offering its necessary information such service type and access point.
b) There is, at least, one agent that provides yellow page service to enable agents find each other.

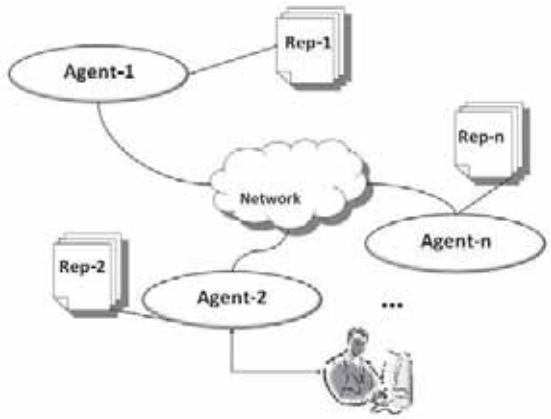


Figure 2. System overview

GAIA design methodology [10] is used to design the MAS. Each individual agent currently holds 4 roles: *Document Annotator*, *Concept Learner*, *Register Handler*, and *Concept Manager*. IBM's UIMA [7] is used to enable dynamical annotation of documents, and, therefore, enable classification of documents within their own repository. In a later phase, we will utilize an addition *Peer Finder* role to create an open cooperative MAS that enables agents automatically find each other. Details of the annotator and learner roles are explained below.

3.1. Document Annotator

Regular search usually is capable of finding tokens without any relationship between them. The tokens are not able to reflect domain-specific properties similar to atoms in early chemistry which are unable to retain chemical properties of a complex chemical substance.

The documents annotator in this project is aiming at annotating "molecules", special combinations of tokens, on which some well-defined constraints are applied. Creating such type of annotation, especially dynamically creating annotations is a fundamental role, not only for concept learning, but also for semantic search involving newly learnt concept. As the following formula describes,

$$Annotation = F_{constraints}(token1, token2, \ldots).$$

Constraints can be, for example, window size, appearing order within certain window size of text, etc. The annotation process is depicted in Figure 3. The methods used in the annotation process are:

CreateConceptHierarchy([concept], keyword1, keyword2, …, CH1)
Annotator first creates a Concept Hierarchy (CH) using received series of keywords. This CH directly goes into the Annotation Engine (AE) to tell what is needed to be searched from the document repository.

CreateAnnotationEngine (Type1, AE1)
This method takes CH as a parameter to dynamically build Annotation Engine (AE) which is the algorithm's container.

DoAnnotation(Annotator1, Doc1, Doc2, …)
Once annotation engine is created, it will be run against repository to annotate and grab satisfying documents.

ReplyQuery(Annotated Documents, PositiveExamples, NegtiveExamples)
This method takes charge of replying to query. In this project, it also implements some specific filtering work such as selection of positive examples and creation of negative examples (see Section 3.2 for details).

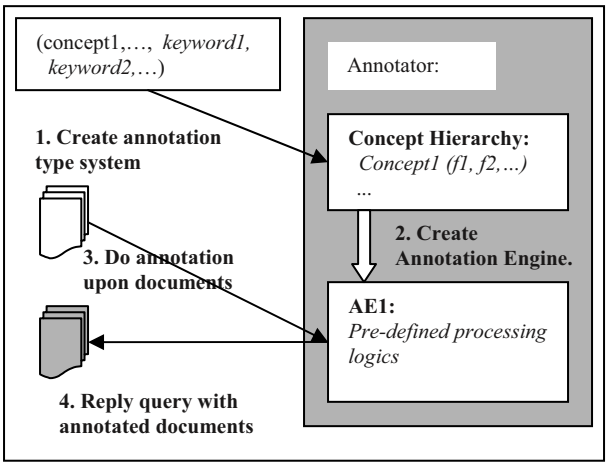


Figure 3. Process of Documents Annotation

3.2. Concept Learner

An agent knowing a concept is equivalent to having a defined classifier for that concept. The classifier is a binary function which decides whether a new incoming document belongs to (or explains) the concept. The *Concept Learner* role is used to generate this classifier.

We adopt a feature-based representation model to represent concepts. That is, each concept is composed of a set of features or keywords which are regarded as the best representation of this concept. For each concept, there exists a set of documents whose major topic belongs to this concept. Those documents are called positive objects for this concept. The list of features can be generated from the positive objects using a simple statistics method [11]. The similarity between a pair of concepts is defined as:

Sc_1,c_2: Similarity between concept c_1 and concept c_2

$Nc_1 \cap c_2$: Number of documents that belong to both concept c_1 and concept c_2 (with relevance degree greater that a defined threshold)

$Nc_1 U c_2$: Number of documents that belong to either concept c_1 or concept c_2.

Based on this definition for each concept set we have an ontology matrix as shown in Figure 4. The values in this matrix denote the similarity value between pairs of concepts. For example, the similarity value between c_1 and c_2 is 0.8, and there is no relationship between concept c_1 and c_3 since the similarity value between them is zero.

	c_1	c_2	c_3
c_1	1	0.8	0
c_2	0.8	1	0
c_3	0	0	1

Figure 4. Ontology Matrix

The learner is based on the training set available. For each concept, we select a set of positive objects (documents in this case) belonging to the concept and a corresponding set of negative objects using the ontology matrix. For example, if we want to create a classifier for concept c_1, we choose positive examples from the documents assigned to concept c_1. For negative examples, we choose documents which belong to concept c_3 because those documents do not represent concept c_1. This mechanism may not always lead to an optimum learning and we have investigated other algorithms [2]. Finally, using the positive and negative examples, we adopt a data mining algorithm to train and get the classifier for concept c_1.

3.2.1. Concept Learning Process

We illustrate the learning process using the following actions (see Figure 5).

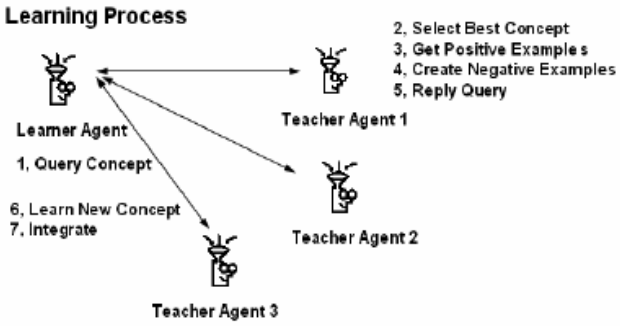


Figure 5: Concept Learning Process

QueryConcept ("keyword1", "keyword2", …)
The learner agent will start the concept learning by issuing action *QueryConcept* which will send the query to other agents. The parameters it takes are a list of keywords representing features of a potential concept.

SelectBestConcept ("keyword1", "keyword2", …)
On the receiver side, the agent first uses keywords to build annotator dynamically which then is used to annotate the candidate documents.

SelectPosEx(concept)
After getting the best concept and candidate documents associated to this concept, the receiver agent will select a given number of positive examples from the candidate documents.

CreateNegEx (concept)
The receiver agent also performs this action to produce a given number of negative examples for the concept based on the ontology matrix.

ReplyQuery(pi, ni)
The receiver agent i sends back the positive (p_i) and negative (n_i) examples to the learner agent.

Learn ((p1,n1), (p2,n2), …)
The learner agent will take all the documents transferred from the teacher agents as the training documents to form a new concept. We have examined various data mining algorithms such as Naïve Bayes and SVM. If there are any conflicting documents, they are dismissed. It means only documents which are agreed by all the agents are regarded as the documents under the new concept.

Integrate (concept)
With the new concept from the method *Learn*, the learner agent will assign a temporary name to it and suggests it to the administrator of the learner system. The administrator can approve the concept, assigns a meaningful name to it and add it to the local repository.

4. IMPLEMENTATION

4.1 System Architecture

Figure 6 shows the prototype system. Document Annotator is developed using IBM's UIMA (Unstructured Information Management Architecture) [7]. IDE Eclipse Europa is selected because it supports UIMA and Apache Tomcat http://www.eclipse.org/europa/) which is used to deploy document annotation service in this project. UIMA is featured by *type system* in which the data has a type and a set of *attribute*, *value* pairs [7], so that it is conceptually identical to concept from our point of view.

4.2. Document Annotator

The components to fulfill actions C*reateConceptHierarchy, CreateAnnotationEngine, DoAnnotation,* and *ReplyQuery* need to be built to form a complete Annotator. Central task is to deal with creation of type system.

4.2.1. Creation of Type System

According to type system definition specification [7], the first step is to build XML-based type system descriptor. Figure 7 shows an actual descriptor from the project. This is an aggregate type system which is composed of several basic primitive types. The lines enclosed by an oval represent one type definition which includes name, description, super-type name, and its features. UIMA provides APIs to build class components to dynamically adjust contents of descriptors and create a corresponding Java class.

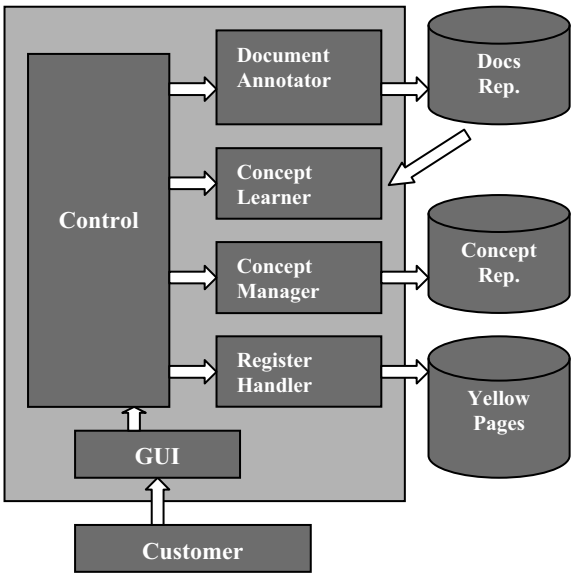


Figure 6. Architecture of Prototype System

4.2.2. Creation of Annotation Engine (AE)

Developer of an annotator has to implement a standard interface having several methods, such as *initialize()*, *process()* and *destroy()* to embed processing logic into it.

There are two ways for creator to tell *process()* method which types need to take and which types need to produce. One is manually creating a component descriptor which is XML-based document, like type system descriptor; another is using APIs that come along with UIMA to dynamically set this component descriptor. The latter is preferable because developer can select input/output type system at run-time. Immediately after the Annotation Engine is created, the *process()* will take over to scan documents and produce types as specified in AE descriptor.

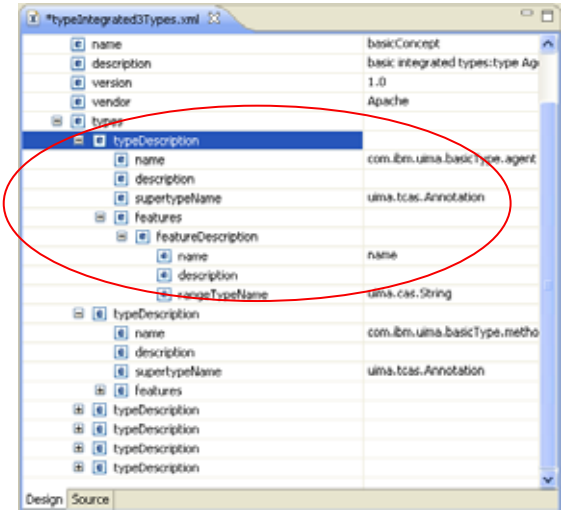


Figure 7. A Type System Descriptor

4.3. Concept Learner

The *Concept Learner* role is responsible for implementing action *Learn* which takes training documents as input, and produces concept classifier.

4.4. Concept Manager

Concept Manager role helps agent to manage set of concepts coming from three sources: newly learnt through training, selected by experts of specific domain, and newly learnt through semantic search. It need implement actions: *SaveConcept, RetrieveConcept.*

5. ILLUSTRATIVE EXAMPLE

To illustrate semantic search-learning process, here we provide a simple but illustrative example. In this scenario, three existing repositories of documents concentrating on software testing (R_{test}), object-oriented analysis and design (R_{oo}), and agent-based software engineering (R_{agent}) are created. Agent-based software engineering (R_{agent}) is considered as the local repository.

In the local repository there are several documents: some of them are about MAS methodology; some about other concepts. The initial status, as shown in Figure 8, tells that all documents, having not been annotated, are organized in flat structure.

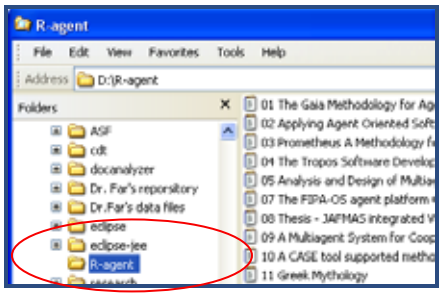


Figure 8. Snapshot of Initial Repository

Based on current status, a regular query, as shown in Figure 9, is initiated. In this case, a user wants to know "what the token Prometheus means in software engineering". Processing this query with no semantic search (as depicted by empty "Concepts" box in Figure 9) will return two documents, with completely different contents. The document 03 is talking about Prometheus MAS design methodology; meanwhile, the document 11 is about Greek mythology which is apparently not the one that user intended to receive.

To disambiguate search results, the agents will take the following steps to kick off a semantic search:

1. A concept learning routine is started to evaluate returned documents, through which new concepts, for example, "agent" (including its features) is identified; it then will be propagated within the group (see [11] for details).

2. Each agent, upon receiving this concept, will annotate its own repository. Annotation procedure re-categorizes repository by conforming to concept hierarchy. Figure 10 shows changes happened to repository R-agent.
3. New concepts will be added to the concept repository in order to support decoding query phrase, or searching.

Figure 9. Illustration of Regular Search Procedure

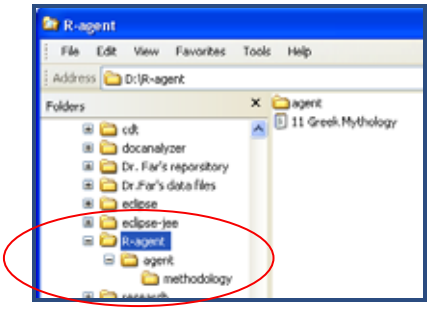


Figure 10. Snapshot of Annotated Repository

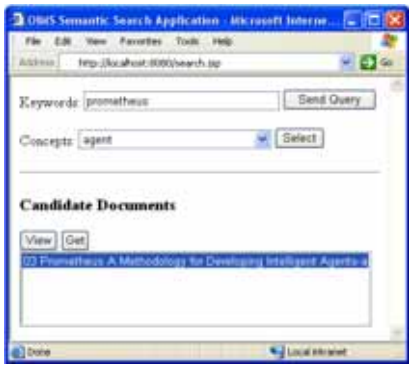


Figure 11. Illustration of Semantic Search Procedure

Once these steps are completed, a next round of search will start by involving the newly learnt concepts. Figure 11 shows a disambiguated result by sending query phrase consisting of both keywords and concepts.

From the procedure explained above, we can conclude that dynamical annotation guided by concept hierarchy is capable of categorizing the repository, consequently, making retrieval of documents more efficient.

6. CONCLUSION AND FUTURE WORK

In this paper we presented a method and a prototype MAS for semantic search-learning. This method is based on the architecture of layered semantic interoperability. The central procedure is composed of dynamical document annotation and concept learning mechanisms to solve the problem of semantic heterogeneity in distributed information management with minimum overhead and no need to commit to a common ontology. From the implementation, we can conclude that the semantic search supported by concept learning is a generative evolutional procedure. It is started by a regular search leading to learning a new concept. Then later the learnt concept is used in semantic search, and the search resolution will improve.

Future work includes implementation of the role *PeerFinder* which will lead to an open MAS. Also, research efforts will be put on organizing concepts into a hierarchy through learning. Finally, based on research achievements, we will propose a protocol for search on other layers of the semantic interoperability model.

7. REFERENCES

[1] M. Afsharchi, B.H. Far, J. Denzinger, "Ontology Guided Learning to Improve Communication among Groups of Agents," Proc. AAMAS'06, pp. 923-930, 2006.

[2] M. Afsharchi, B.H. Far, and J. Denzinger, Learning Non-Unanimous Ontology Concepts to Communicate with Groups of Agents, Proc. IAT'06, IEEE Press, pp. 211-217, 2006.

[3] S. Decker, S. Melnik, F. Van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, I. Horrocks, "Semantic Web: The Roles of XML and RDF", *IEEE Internet Computing*, 2000.

[4] J. Euzenat, "Towards a principled approach to semantic interoperability," A. Gomez-Perez et al (eds.) IJCAI'2001 Workshop on Ontologies and Info Sharing, Seattle, 2001.

[5] B.H. Far, A.H. Elamy, N. Houari and M. Afsharchi, "Adjudicator: A Statistical Approach for Learning Ontology Concepts from Peer Agents," Proc. SEKE'07, 2007.

[6] R. Guha, R. McCool, E. Miller, "Using the semantic web: Semantic search," Proceedings of the WWW'03, 2003.

[7] IBM, "Unstructured Information Management Architecture (UIMA)", http://domino.research.ibm.com/comm/research_projects.nsf/pages/uima.index.html, 2007.

[8] K.C. Jim, C.L. Giles, "Talking Helps: Evolving Communicating Agents for the Predator-Prey Pursuit Problem," *Artificial Life 6(3)*, 2000, pp. 237–254.

[9] A.B. Williams, "Learning to Share Meaning in a Multi Agent System, Autonomous Agents and Multi Agent Systems 8(2)," pp. 165–193, 2004.

[10] N. Wooldridge, and D. Kinny, "The GAIA methodology for Agent-Oriented Analysis and Design," 2000.

[11] Y. Zilan, C. Zhong, B.H. Far, "A Practical Ontology-Based Concept Learning in MAS," Proc. IEEE CCECE'08, (to appear), 2008.

Automating a domain model aware reengineering methodology

Javier Belmonte
University of Geneva
Geneva, Switzerland
belmont2@etu.unige.ch

Philippe Dugerdil
HEG-Univ. of Applied Sciences
Geneva, Switzerland
philippe.dugerdil@hesge.ch

Abstract—Program comprehension's importance as a research field has recently increased. In fact the task of understanding the working of a program roughly represents half of the maintenance costs. Our approach to software comprehension is to map the high level models of the system to the source code. The building of the models follows the technique proposed in the Unified Process (UP). The model recovery process, that has been presented elsewhere, rested so far on manual techniques. But to offer a real help to software maintainers, it should be automated. This papers presents the tools we built to automate the mapping of high level business and system models to the source code. These tools use artificial intelligence techniques to make heuristic search of the possible maps between the models. After having explained the way we solved the problem, the paper shows a comparison of the performance of the tool as compared to the manual application of the process on a real industrial software.

1. Introduction

It is commonly known that maintenance is the most expensive part in the life cycle of a software system; it represents between 60% and 90% of the total cost of a program [8]. It is less commonly known (yet extremely interesting) that almost two thirds of the maintenance cost are devoted to software comprehension or understanding [16]. The importance of program comprehension in maintenance comes from the nature of reengineering. In fact, before being able to restructure or re-implement a legacy program it is necessary to understand its working and its functional architecture [2]. During the process of implementing a program, developer's understanding of the program grows since they have to keep in mind which elements of the system satisfies what part of the specifications. Unconsciously, developers build a mental mapping between the problem domain elements (business domain or real world) and the elements of the system domain (source code) [20][23]. This mapping represents their understanding of the program being developped. This mental process agrees with the following definition of program comprehension [1]:

"A person understands a program when able to explain the program, its structure, its behavior, its effects on its operational context, and its relationships to its application domain in terms that are qualitatively different from the tokens used to construct the source code of the program."

The rebuilding of these mental mappings seems then necessary for program comprehension. It allows understanding of the system domain components by linking them to business domain components. The surprising fact is that most of the remarkable techniques for program comprehension or reengineering fail to take domain elements into account and focus mainly on the syntactical features of the programs [3].

A reengineering methodology, relying as much on the analysis of the properties of source code as on the existence of the mapping between the domain and the code is proposed in [3]. This methodology allows the construction of a "bridge" between the source code of a program and its analysis diagram (robustness diagram) as reconstructed from the specifications. It is important to note that this methodology has been empirically tested on a real life legacy system [14] leading to the next natural step: automating. This has been done using artificial intelligence and knowledge engineering techniques. This article focuses on these techniques and the results that have been obtained.

First, a brief reminder to our reengineering methodology is given in section 2. Then, Sections 3 and 4 focus on the system that has been built to automate the methodology. In section 5, we discuss the results comparing them, when possible, to the ones obtained manually by Jossi on the same system [14]. Section 6 presents the related work in domain-driven reverse-engineering and section 7 concludes the paper by explaining the future work.

2. Reengineering methodology

The methodology presented in [3] is based on the development process know as Rational Unified Process (RUP) [15]. Its goals are:

- To rebuild the domain and system models that a developer may have used during the forward engineering of the program.
- To construct a mapping between these models and the actual code of the system.

To tackle the reengineering problem in its greatest generality, the methodology considers the documentation of

the program as nonexistent and its original developers as unavailable. The lack of these two information sources forces the methodology to focus on the current state of the program and its actual users. In fact, even if the actual users of the program ignore everything about the implementation of the program, they represent an invaluable source of information on the business purpose and the features of the system. Then, the methodology suggests that actual users be consulted in order to gather information useful to rebuild the domain and system models.

Therefore, our reengineering methodology can be split in two phases, each one focusing in one of the two above-mentioned goals. To ease global comprehension, we refer to the normal way the RUP phases are developed as the "forward" direction (this is represented by the gray arrow in figure 1):

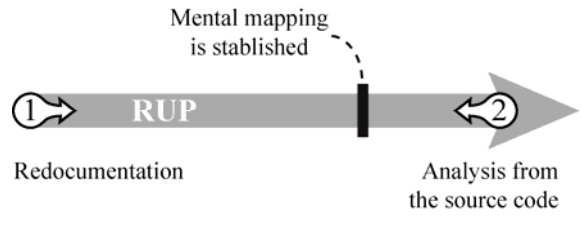


Figure 1: The two phases of our methodology.

The first phase focuses mostly on the re-documentation of the functional specifications of the program and their analysis. It is developed in the "forward" direction of RUP. In the second phase, we rebuild a mapping between analysis models and the source code of the programs. This is performed "backwards". The moment in the RUP timeline where these two phases meet might be seen as the moment in time where the mapping we're trying to rebuild was established by the original developers of the system. It is also the moment in time at which the maintainers of the program reestablish this knowledge.

A. First phase

The first phase focuses on re-building the business and system models. This ends up with the System Use-Case Model and the associated Analysis (Robustness) diagram. It follows the disciplines of the RUP:

- Reconstruction of the business models, Business Use-Case model and Business Analysis model.
- Reconstruction of the system models, the System Use-Case model and the Analysis (Robustness) diagram associated to every use-case.

The analysis diagram is built from the following analysis objects stereotypes [15]: Boundary (interface to the external world), Entity (information holder) and Control (orchestrator of the use-case's execution). Then, the above mentioned models allow us to establish the traceability links between the functional needs at the business level and the analysis objects that represent the corresponding system responsibilities.

B. Second phase

In this phase we concentrate on the mapping between the system models, especially the robustness diagram and the source code elements. By re-creating the links between the analysis objects and the classes of the programs we are able to close the traceability link from the high level business models and the implementation [4]. To create these links we use two kinds of techniques:

- **Static analysis:** this works on the static structure of both the source code and the robustness diagram. They are called static because they rest on the syntactic features of the elements.
- **Dynamic analysis:** this works on the features of the system that can only be observed at run-time. Most of this information concerns the interactions between the elements of the program, their frequency, their nature, and their order. By executing the system we will compare what actually happens inside the system with what was supposed to happen (as documented by the system models). We will then search for similarities that lead to the mappings.

We perform post-mortem dynamic analysis i.e. we generate an execution trace file during the execution of the system that we analyze off-line. One of the problems of dynamic analysis is to choose the scenario to execute on the system i.e. what action should the user perform for the execution trace to convey the required information [9]. Because of our approach, we concentrate on the recovered use-case of the system that represents what the users actually do with the system. Then, we gathered an execution trace for every use-case flow we recovered from the first phase of the methodology. This execution trace holds a sequence of operation invocations whose order is obviously *correlated* to the user manipulations. This observation is central to our reengineering methodology.

3. The platform

Three characteristics of both our methodology and the problem we are trying to solve had a strong impact on the way we implemented the platform:

1. We have described our approach to program comprehension as a forward and backward application of the RUP. Then, according to RUP, our methodology is iterative and incremental as well [4].
2. Most of the information used to rebuild the program and system models are gathered from the users. So, we cannot be sure of their accuracy.
3. Human reasoning is a logical process in which new information is obtained by evaluating previously known information [22]. Therefore, the platform should allow the revision of the old inferences in the light of new information.

By the way, it is interesting to note that the iterative nature of the RUP is at the same time necessary to the simulation of human reasoning. Indeed, obtaining new

information can be seen as one iteration of the simulation. Therefore, at each iteration, new information merges into the group of known information. These information units will be called *facts* from now on.

So far, it could seem that a production system (i.e. the artificial technique to simulate reasoning by production rules [17]) is all we need to perform the simulation. However, such a production system could not easily deal with uncertain facts. Then, our system should be able to revise its reasoning while new facts are generated by the production system. Consequently, we chose to adapt a TMS (Truth Maintenance System) [7] to control the production system as well as the set of facts produced and stored at each iteration. The TMS works with a Knowledge Base (KB) that holds all the facts generated by the production system. As we said, human reasoning generates new facts in an iterative fashion. This process could be decomposed into several smaller and simpler reasoning units which, together, perform the whole reasoning. These smaller reasoning units are called *inference rules* in the TMS paradigm.

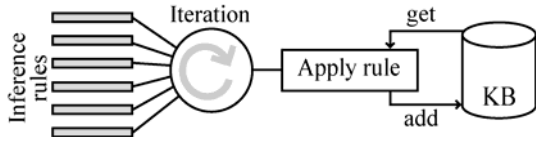


Figure 2: Our system components and process.

Figure 2 shows a representation of the elements composing our system: the KB and the set of inference rules. Its execution can be described as follows: the inference rules are applied iteratively on some of the facts in the KB, the facts produced by the inference rules are added to the KB at the end of each iteration. Then the TMS engine is responsible for assigning a level of certainty to each fact in the KB.

4. Mapping analysis objects to source code

There are two sources of information that are used to perform the search of the mappings:

1. The description of the use-case flows.
2. The execution trace gathered while executing the scenarios corresponding to the use-cases.

In fact, these are two representations of the same reality: the performance of a business function by the system. The first is located at the user level, the other at the system level. Since the operations of the system are called in the same order as the manipulation of the user, we should be able to correlate both representations. Use-case flows normally consist of a sequence of text-lines (steps) expressing the interaction of the user with the system and the response of the system. The information in this representation is not formalized thus leading to subjective interpretations. On the other hand, execution traces are made up of a sequence of operations called during the execution of the use-case, together with their actual parameters and returned values, making no room for misinterpretation. We considered that there were little chances to succeed in comparing both representations without first adding information to the use-case flows that would bring them closer to the elements observed in the execution traces.

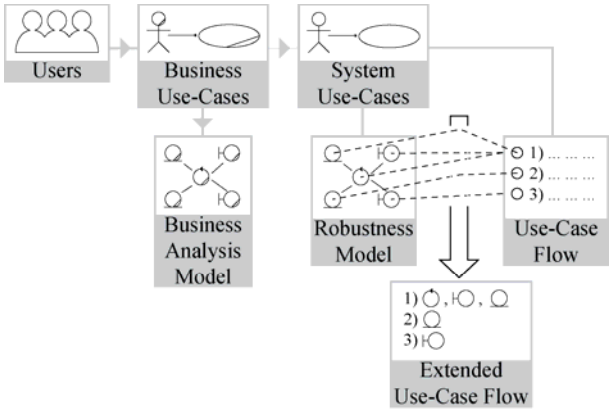


Figure 3: Building the extended use-case flows

Robustness diagrams are built by analyzing the use-case flows and determining which robustness diagram objects are involved at each step [15]. Most of the times, this mapping is not documented and generally lost because there is no need to keep it at the end of system development. In our case nevertheless, this relationship between use-case flow steps and analysis objects is exactly what we need to re-create. We then built the *extended flows* for each use-case containing, for each step, the list of the involved analysis objects. Figure 3 shows the process of building the extended flows.

Finally, from a close examination of the execution trace we realized that some of the modules are invoked all over the timeline of the execution. Since no objects exhibit such a behavior in the extended flow of the use-cases we pre-filtered the execution trace by removing these "omnipresent" modules since they are unmatchable.

A. Mapping the boundary objects

Boundary objects represent interfaces between the outside world and the system. The matching technique is to compare the positions of the occurrences of the boundary objects in the extended flow of the use-cases and in the execution trace. We should then be able to identify which source code components are likely to implement what boundary object. However, we cannot expect to observe an exact match, mostly because the density of occurrences is extremely different between the extended flows of the use-cases and the execution trace. Indeed, while occurrences in extended flows are a small multiple of the number of steps in a use-case flow, which stays generally under 50, the number of occurrences in a execution trace can easily count in thousands.

To bring the two descriptions close enough to find matches between their elements, two strategies are developed:

1. Bring down the number of occurrences of elements to analyze in the execution trace by filtering those that cannot implement a boundary object, based on syntactical considerations.
2. Summarize the multiple occurrences of a given element in the execution trace with a function computing the "density" of occurrence. This function allow us to bind an occurrence of the boundary object in the extended flow with the element of the execution trace that has the highest density of occurrence at the same moment in time.

Before talking about the actual implementation of these strategies, we need to clarify some of the terms used in our experimentation. The system on which our technique has been tested is an industrial package software. It is a fat-client kind of client-server system. The client is made of 240k lines of VB6 code. The server consists of 80k lines of PL/SQL code accessing an Oracle database. In this paper, for the sake of conciseness, we will concentrate on the reverse engineering of the client part of this system. The elements of the execution trace will be Visual Basic procedures invoked during the execution of the UC flows. These procedures belong to modules which represent the level of granularity we target in our mapping process: the analysis objects are mapped to VB modules.

The first strategy was performed manually by Jossi [14]. He manually selected, based on the extension of the file name, the modules in the source code that implement a window. This technique was easily automated on our platform, reducing from 363 to 12 the number of modules to look for in the trace file and from 27537 to 1954 the number of events to analyze in the execution trace. This filtering step has then become a proper inference rule in the production system. We now have to look for the mappings between the remaining modules and the boundary objects in the extended flows of the use-cases.

Our technique is to compute a measure of closeness between a module occurrence in the execution trace and a boundary object occurrence in the extended flow. First, we normalize the timeline of both descriptions, the extended flow and the execution trace, to the interval [0..1]. Every event will then take place in this normalized timeline. The closeness measure is not only computed on the base of the distance between occurrences but also on the following concepts:

- **Depth of the call:** since the robustness object in the extended flow of the use-cases are considered the key object implementing the behavior they should be readily visible in the execution trace. In other words they should not be deeply nested in the execution call chain. Then, we implemented the following heuristic: the deeper the invocation in the call chain, the less probable the match.
- **Coverage:** represent the extent of use of a module along the timeline. The more the spread of the occurrences of a module the larger the coverage. Therefore we favor modules whose occurrences are located in specific moment in time, over modules whose occurrences are evenly distributed along the timeline.

Finally, our closeness metric c is computed by the following formula where d, p and v represent distance, depth and coverage respectively:

$$c = \frac{1}{d}\left(\frac{1}{p}\right)^{K_1}\left(\frac{1}{v}\right)^{K_2} \qquad (1)$$

The values K_1 and K_2 have not yet been formally investigated so far. In our implementation, we empirically determined the best values by trial and error. The closeness formula computes the match between a single occurrence of a boundary object in the extended flow and an occurrence of a module in the execution trace. Next we must compute the match over the entire timeline of the execution of the use-case. In order to display the result as a graphic, we compute the moving average of the closeness value over the timeline. The result is presented in figure 4. The resulting value is called the "significance level" of a module to a boundary object. In this figure, the vertical axis represents the significance level of a particular module at the in time represented by the horizontal axis.

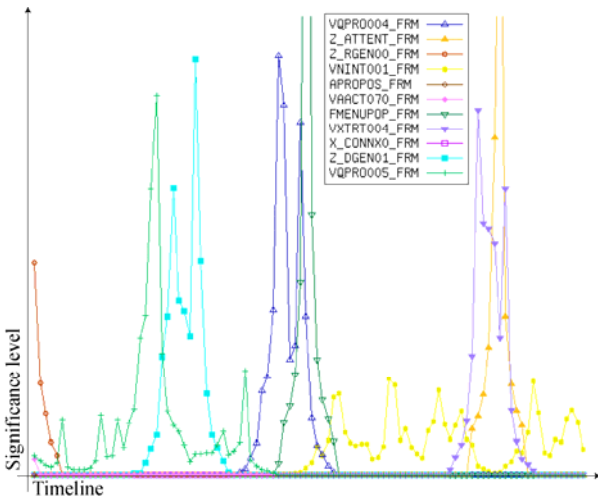


Figure 4: Significance levels for modules.

To establish the final mapping, we took each boundary object and computed, for every module, the mean of the significance value for each of the objects occurrences. We normalized the results for each boundary object so that their

sum was 1. This way we gave more importance to the high significant modules without overlooking the significance of other modules. The normalized values gave us a meaningful way of ordering the mappings between each boundary object and each module according to their likely match.

B. Mapping Entity objects

A first analysis of the modules used to access the tables has shown that all access went through the same few modules. In other words, there is no specificity of the modules to the entity objects of the robustness diagram. Therefore we decided that it would be more useful to re-create the mapping between the entity objects and the database tables that store the information represented by these objects. As before, we compared the occurrences of the objects in the extended flows of the UC and the table access in the execution trace. Then, we need to analyze the values of the parameters of the operations in the execution trace to find the strings corresponding to SQL queries. This activity has been automated by another heuristic rule.

Nevertheless, the mapping heuristic is different from the one we used to map boundary objects because:

1. The mapping of an entity object to a table can be observed as one global access or as many localized accesses. Therefore, coverage does not play any role in this mapping.
2. Unlike operations, there is no hierarchy of accesses to a table. Therefore, there is no equivalent to the depth of calls in the case of table access.

Distance between occurrences in the normalized timeline is the only measurement that we use in this rule to compute the closeness of the match. However, the mapping is much fuzzier in the case of database tables than modules because:

- The same table might be used to store several information items belonging to different entities.
- A table could be used in read mode to validate information stored in another table.

Therefore we decided to manage the match between the entity object occurrences and the table occurrence like the symptoms and illnesses in medical problem solving. In other words, the closeness between the object occurrences in the timeline and the table occurrence in the timeline is considered the symptom and a true correlation between both items (object and table) as the diagnosis. Then, we measure the specificity and sensibility of the relationship between the symptoms and the diagnosis as a way to evaluate the strength of the match. We proceed in two steps: first we compute the sensitivity then the specificity. To formalize the relationships between "symptoms" and "diagnosis", we use the following definitions:

- {T,O} represents the mapping between the table T and entity object O (i.e. the diagnosis)
- T | O represents the closeness between the positions of T and O on the timeline (i.e. the symptom). For the closeness to be true, the distance between the occurrence of the object and the table should not be larger than 5% of the timeline. We estimated empirically that a bigger distance would not let us to conclude anything.

Figure 5 presents the truth table for the closeness operator. We shall remark that the function is not defined in the case where p and q are both absent.

p	q	p|q	¬(p|q)
0	0	?	?
0	1	0	1
1	0	0	1
1	1	1	0

Figure 5: Truth table of the function of closeness $p|q$.

1st step: sensitivity: it is computed as the probability to observe a short distance between the table and the entity objects if these elements are truly correlated:

$$\{\mathrm{T},\mathrm{O}\} \rightarrow (\mathrm{T} \mid \mathrm{O})$$

In order to approximate this probability, we made the following supposition: the higher the probability of $(T \wedge \neg O) \vee (\neg T \wedge O)$, the less likely $\{T,O\}$. In other words: $(T \wedge \neg O) \vee (\neg T \wedge O) \rightarrow \neg\{T,O\}$

However, since $(T \wedge \neg O) \vee (\neg T \wedge O) \equiv T \oplus O$ and given that the meaning of $\neg(T \mid O)$ is $T \oplus O$ [1] (figure 5) we have: $\neg(T \mid O) \rightarrow \neg\{T,O\}$, which is equivalent to equation (2). This means that $P(T \oplus O)$ is the likelihood of $\neg\{T,O\}$.

2nd step: specificity: it is computed as the probability to not observe a short distance between the table and the entity objects if these elements are not correlated

$$\neg\{T,O\} \rightarrow \neg(T \mid O)$$

Or equivalently: $(T \mid O) \rightarrow \{T,O\}$

In order to compute this probability we assumed the following: the more probable $(T \mid O)$, the more likely $\{T,O\}$. Within this assumption, the probability of $\{T,O\}$ can be approximated by probability of $(T \mid O)$, which is the probability of $(T \wedge O)$ (figure 5).

By combining these two metrics we can build a formula that, applied to every couple (table; entity object), gives us measure of the likelihood of a true correlation between them.

Let P_1 be the value of $P(T \oplus O)$ (sensibility) and let's P_2 be the value of $P(T \wedge O)$ (specificity). We combine these two results through the following equation of the certainty of a match between T and O:

$$c = K \cdot P_2 - P_1 \qquad (2)$$

K is a constant allowing us to give more or less importance to P_2 according to the specific situation at hand.

[1] This holds because we are not interested in the cases where both instances, T and O are absent.

5. RESULTS

In order to evaluate the performance of our heuristic-based matching system we compared its results for the mapping of the boundary objects to the ones obtained manually by Jossi [14]. It must be noted that all the mappings produced by our engine are weighted by the certainty of the match. This is not the case in the manual match performed by Jossi. Therefore, we cannot directly compare a set of weighted matches to individual matches obtained by hand. Then, we compared the most likely matches produced by our engine to each match obtained manually.

Our system produced 14 mappings and the manual match 16. The comparison between both techniques has shown that 13 mappings were the same. This means that:

- Our system was able to correctly establish 81% of the manual mappings.
- From the 14 selected mappings made automatically, 93% were correct.

	Boundary object	Modules mapped by Jossi	Modules mapped by our system
1	Create a new folder	VQPRO005.FRM	VQPRO005.FRM
2	Context	VQPRO005.FRM	VQPRO005.FRM
3	Persons	VQPRO005.FRM Z_RGEN00.FRM VINDI001.FRM	VQPRO005.FRM
4	Address	VQPRO005.FRM Z_DGEN01.FRM	VQPRO005.FRM Z_DGEN01.FRM
5	Address input	VQPRO005.FRM Z_DGEN01.FRM	VQPRO005.FRM Z_DGEN01.FRM
6	Folder explorer	VQPRO004.FRM	VQPRO004.FRM
7	Step management	FMENUPOP.FRM	FMENUPOP.FRM
8	Evaluation	VXTRT004.FRM	VXTRT004.FRM Z_ATTENT.FRM
9	Modalities	VNINT001.FRM	VNINT001.FRM
10	Intervention decision	VNINT001.FRM	VNINT001.FRM
11	Characteristics	VNINT001.FRM Z_ATTENT.FRM	VNINT001.FRM

Table 1: Comparison of the manual and automatic mappings.

Table 1: Comparison of the manual and automatic mappings. shows the mappings that were obtained by Jossi and by our automated system. At the same time, it shows, by highlighting them, the mappings that are different between the results of both techniques.

The mapping between tables and entity objects was not performed by Jossi because of the high quantity of information to process. Jossi limited his investigation to the search of the modules that performed the accesses to the database. It is not possible then to compare the results of the inference engine with any previous results, as we did in Table 1. Therefore, to analyze these results, we will compare them to our expectations:

- A mapping should be created between tables and objects whose occurrences are *often* close on the timeline.
- A mapping should not be created between tables and objects which are *often* far from each other on the common timeline.

Figure 6 presents the results of the mapping of entities to tables. The vertical axis represents the different tables. The horizontal crosses shows the moment in time where the table is accessed. The horizontal axis represents the timeline on which we set the occurrence of the entity objects. For example {entity1,T4} and {entity2,T3} satisfy the first expectation. The candidate mapping {entity2,T4} is rejected because their occurrences are close on one part of the timeline only (about 0.6 in the normalized timeline). In fact they are more often far from each other than close.

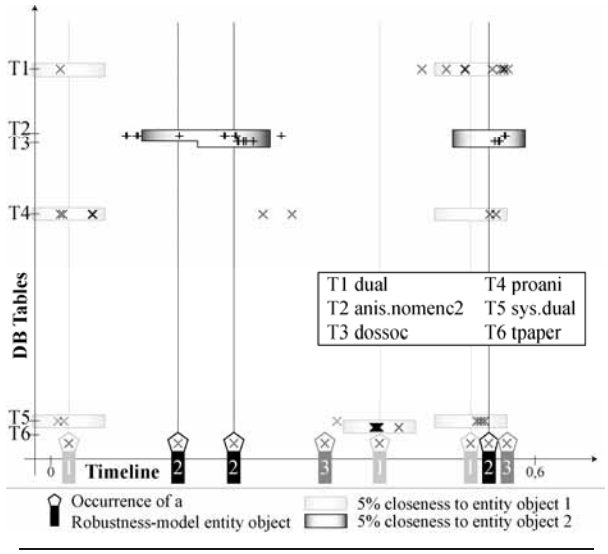


Figure 6: Results of the mapping of entities to the tables.

The tolerance to non-matches is controlled by the parameter K. This let us keep {entity2,T2}, although there are some non-matches along the timeline, but reject {entity3,T5} because:

- T2 occurrences didn't happen close to the occurrences of any other entity object than entity2. Therefore, the mapping was a lot more likely for entity2 than for the other entities.
- T5 occurrences are close to more occurrences of entity1 than entity3. Therefore, T5 was mapped with entity1 with a higher value of certainty than with entity3.

6. RELATED WORK

Domain models have long been acknowledged as a good way to improve reverse engineering and program understanding [19][18]. For example, in their Pioneering work, DeBaud and Rugaber [6] and DeBaud [5] used an executable domain model in the form of an object oriented framework. This framework represents the concept of the domain and helps the search for the corresponding concept in the programs. Then, each time a concept is identified in

the program being reengineered, the corresponding class of the domain is instantiated representing the result of the reverse-engineering [6]. From this experiment, DeBaud [5] remarked that domain models can efficiently guide software artifact comprehension. However, their structure is somewhat subjective and depends on the application. In other words, in the same domain, many different "models" can be represented. Therefore such a model must be flexible to account for all the variations of the domain concepts used in the program. Moreover DeBaud highlighted the difficulty to match the granularity of both the program code elements and the domain model. In particular he mentioned the difficulty raised by the delocalization of the concept in the program i.e. the lines of code referring to a concept are often noncontiguous. It must finally be noted that the domain model used in this work represents programming concepts not business concepts. This approach is quite similar to the work of Gold [12] who, in its HB-CA system for concept assignment used a knowledge base of programming concepts. In this work also, the domain model does not represent business concepts but programming concepts. For example, the experiment reported refers to programming notions such as "read file", "write file", "compute value". Using this knowledge, HB-CA is able to identify the segments of the source code where files are written, read etc. Later, the problem of non-contiguity of the source statement dealing with a concept was solved by Harman, Gold et al. by using slicing techniques [13]. Rugaber and Stirewalt used a formal specification using an algebraic specification language to model both the domain and the program being reverse-engineered [18]. Their main purpose is to evaluate the effort needed to reverse-engineer a program. Then they build the model of the program and, using a code generator, they generate a new system from the formal specification. If the generated system is "close enough" to the original system, the model of the system is considered adequate. They can afterwards evaluate the reengineering effort based on this model of the program. However, to construct it, they first build a domain model that gives them expectations for the concepts that might be represented in the code. Then they build the program model. Next they link the program model to the domain model, an operation called "interpretation". These connections help to understand the system purpose and how the code fulfils that purpose. The approach taken by Gall, Weidl and Klotsch [10][11][24] is to build an object model of the main domain abstraction implemented in the code, using all documentation and expert advice available. This is considered the domain model. Then the source code is analyzed to find candidate software objects. Finally the two sets of objects are matched using a similarity factor based on syntactic (name) and semantic (data type) properties. Human expertise is required to solve the many ambiguities that arise during this process. Their experiments have shown that the amount of code that the system could match against some domain concepts is limited. In fact many program elements are left without finding a suitable mapping. The conclusion we can draw from this work is that the code associated to the entity objects in a given program only represents a limited part of the total amount of code.

7. CONCLUSION

The heuristic techniques we used to automate the match between the objects of the robustness diagram and the programming elements produced very encouraging results. The main findings are:

1. The manual match between the objects in the robustness diagram and the elements of the implementation of the system can be automated.
2. The result obtained show that the automatic match can outperform the manual match especially if the quantity of information prevents any manual match to be done.
3. The implementation based on a production system coupled with a TMS is a workable solution. In particular, the processing time is reasonable with respect to the amount of information to process. In particular the processing of a trace of more than 25'000 events took less than a dozen of seconds in all our experiments.

This system has been developed in Java as an Eclipse plugin. It has been linked to a UML modeler so that the re-constructed models of the program can be used as input to the heuristic matcher. Future work ideas include:

- Extend the mapping engine to deal with higher abstraction models built with the RUP methodology like the Domain Object Model representing the key abstractions in the domain.
- Extend the specificity / sensibility metrics to all domain model objects.
- Improve the certainty calculation to better take into account the iterations between several use-cases and their partial matches.

8. REFERENCES

[1] Biggerstaff T.J., Mitbander B. G., Webster D. E. - Program understanding and the concept assignment problem. *Communications of the ACM, vol. 37, no. 5, pp. 72–82, 1994.*

[2] Clayton R., Rugaber S., Wills L. – On the Knowledge Required to Understand a Program. *Proc. IEEE Working Conference on Reverse Engineering WCRE'98, 1998.*

[3] Dugerdil P. - A reengineering process based on the unified process. *Proc. of the 22nd IEEE International Conference on Software Maintenance. ICSM 2006*.

[4] Dugerdil P. - Using RUP to Reverse Engineer a Legacy System. *The Rational Edge, September 2006. www.ibm.com/developerworks/rational/rationaledge/*

[5] DeBaud J.-M. - Lessons from a Domain-Based Renegineering Effort. Proc IEEE WCRE 1996.

[6] DeBaud J.-M., Rugaber S. – Software Reengineering method using Domain Models. *Proc IEEE International Conference on Softrware Maintenance (ICSM) 1995*.

[7] Doyle J. - A Truth Maintenance System. *Artificial Intelligence 12(3), 1979*.

[8] Erlikh L. - Leveraging legacy system dollars for e-business. *IEEE IT Professional, vol. 2, no. 3, pp. 17–23, 2000.*

[9] Eisenbarth T., Koschke R. – Locating Features in Source Code. *IEEE Trans. On Software Engineering 29(3) March 2003*.

[10] Gall H., Klosch R. Mittermeir R. – Using Domain Knowledge to Improve Reverse Engineering. *Int. J. on Software Engineering and Knowledge Engineering (IJSEKE), 6(3) 1996.*

[11] Gall H., Weidl J. – Object-Model Driven Abstraction to Code Mapping. *Proc. European Software engineering Conference, Workshop on Object-Oriented Reengineering, 1999.*

[12] Gold N. - Hypothesis-Based Concept Assignment to Support Software Maintenance. *PhD Thesis, Univ. of Durham,UK,2000.*

[13] Harman M., Gold N., Hierons R., Binkeley D. – Code Extraction Algorithms which Unify Slicing and Concept Assignment. *Proc IEEE Working Conference on Reverse Engineering (WCRE'02), 2002.*

[14] Jossi S. - Reverse-engineering du système d'information. *Bachelor thesis, HEG - Univ. of Applied Sciences, Geneva,Switzerland, 2006*.

[15] Jacobson I., Booch G., Rumbaugh J., The Unified Software Development Process. *Addison-Wesley, 1999.*

[16] Muller H.A., Tilley S.R., Wong K. - Understanding software systems using reverse engineering technology perspectives from the Rigi project. *Proc.of the 1993 conference of the Centre for Advanced Studies on Collaborative research. IBM Press, 1993.*

[17] Nilsson N.J. - Principles of Artificial Intelligence. *Tioga Publishing, 1980.*

[18] Rugaber S., Stirewalt K. – Model-Driven Reverse Engineering. *IEEE Software July/August 2004*.

[19] Sayyad-Shirabad J., Lethbridge T.C., Lyon S. – A Little Knowledge Can Go a Long Way Toward Program Understanding. *Proc IEEE Workshop on program Comprehension (WPC) 1997.*

[20] Storey M.A. – Theories, tools and research methods in program comprehension: past, present and future. *Software Quality Journal 14(3), 2006.*

[21] Tilley S.R., Smith D.B., Paul S. - Towards a framework for program understanding. *Proc. IEEE Int. Workshop on Program Comprehension (WPC'96), 1996.*

[22] van der Sraaten S. M., - Human deductive reasoning: a formal logic versus a mental model theory. *MS Thesis, Rand-Africaans University, Johannesburg, South Africa, May 2003*

[23] von Mayrhauser a., Vans A.M. – Program Comprehension During Software Maintenance and Evolution. *IEEE Computer, August 1995*

[24] Weidl J., Gall H. – Binding Object Models to Source Code: An Approach to Object-Oriented Re-Architecting. *Proc. IEEE Annual International Computer Software and Applications (Compsac) 1998.*

Explaining Product Release Planning Results Using Concept Analysis

Gengshen Du, Thomas Zimmermann, Guenther Ruhe
Department of Computer Science, University of Calgary
2500 University Drive NW, Calgary, Alberta T2N 1N4, Canada
{dug, zimmerth, ruhe}@cpsc.ucalgary.ca

Abstract

Objective: This paper aims to generate explanations from a series of data points obtained from a decision support system called ReleasePlanner® for supporting product release planning and considered to be a black box.
Method: Concept analysis is applied to 1085 data points received from running 10 scenarios of a real world product release planning project with 35 candidate solutions generated by ReleasePlanner®.
Results: Three main results are obtained: (1) patterns between inputs and outputs; (2) impact of individual input parameters on outputs; and (3) sensitivity level of outputs in dependence of inputs.
Conclusion: Concept analysis is shown to be a feasible technique for gaining more insight into the structure of results obtained from a black box input-output system, such as, but not limited to, ReleasePlanner®.

Keywords

Explanations, concept analysis, product release planning

1 Introduction

Product release planning involves decision making on assigning features to different releases for incremental software product development. It must simultaneously consider several aspects, such as conflicting stakeholder priorities and objectives, feature interdependencies, and resource and risk constraints [15]. A decision support system called ReleasePlanner® [13] has been developed to support decision makers in the complex release planning process. It is based on computationally efficient optimization algorithms for the generation of a set of alternatives solution having a proven degree of optimality.

However, the findings from a series of experiments conducted with ReleasePlanner® users revealed that they were reluctant to accept the solutions advised by this tool [6]. Similar observation has also been made on other systems [4] [10]. It was concluded in [1] that the major problems are not technical problems, but people problems in which people have very limited understanding on the support they get from decision support systems. In addition, according to [2], product release planning problem is classified as a wicked problem [14] which is hard to be precisely formulated. Thus the procedure needed to solve product release planning problems (as demonstrated in ReleasePlanner®) is more complex and requires more in depth explanations to achieve good user understanding on the tool support and its solutions.

How can we facilitate better understanding of the ReleasePlanner® solutions? In this paper, a data analysis technique called concept analysis [11] is applied for this purpose. It is applied to investigate data within a specific product release planning problem and identifies hidden relationships between the project inputs and outputs. In particular, three types of relationships are analyzed:

- Patterns between the input and output attributes
- Impact of individual input attributes on the outputs
- Sensitivity level of the outputs to the inputs

The answers to these three research questions provide additional knowledge that is currently unavailable to users of the ReleasePlanner® system. As a result, the users' acceptance and trust level on the tool and its solutions is expected to be improved.

The remainder of this paper is organized as the follows. Section 2 gives an overview of product release planning and the related decision support tool ReleasePlanner®. Section 3 introduces the background of concept analysis. In Section 4, a sample product release planning project is investigated to demonstrate the application of concept analysis. Section 5 analyzes and interprets the results in the context of the three stated questions. Finally, Section 6 summarizes the research and outlines future research.

2 Product Release Planning

Many formal approaches have been proposed for product release planning, such as incremental funding method [5], cost-value approach [9], planning software evolution with risk management [8], and hybrid intelligence (EVOLVE*) [15]. The latter is used in this paper. This section gives a short overview of this approach to the extent necessary to understand and judge the results obtained from concept analysis presented later in this paper. More details on the method are available from [15].

2.1 Technical Formulation

In incremental product development, the goal of product release planning is to select from a set of features $F = \{f_1,\ldots,f_n\}$ and to assign them to one of K possible releases each of them having a weight (relative importance) of ξ_k ($k = 1\ldots K$). A release plan is described by vector x with

decision variables $\{x(1),\ldots,x(n)\}$, where $x(i) = k$ if feature f_i is assigned to release option $k \in \{1,\ldots,K\}$; and $x(i) = K+1$ otherwise (i.e. the feature is postponed).

Two types of feature dependencies are considered: coupling and precedence relationship. A coupling $CC(f_i, f_j)$ indicates that both features f_i and f_j must be released jointly. A precedence $PC(f_i, f_j)$ indicates that feature f_i cannot be released later than f_j. Some features can be fixed to certain release by the pre-assignment $preassign\text{-}x(i)=k$, indicating that f_i is fixed to release k.

The planning approach considers T resource types for implementing the features. Capacities $Cap(k, t)$ relate each release k to each resource type $t \in \{1,\ldots,T\}$. Every feature f_i requires an amount of resources of type t $r(f_i, t)$. Thus, each release plan x assigns feature f_i to release k expressed as $x(i) = k$, for all releases k and resource types t, must satisfy $\Sigma_{x(i)=k}\, r(f_i, t) \leq Cap(k, t)$.

A set of stakeholders $S = \{s_1,\ldots,s_q\}$ is involved in release planning. Each of them has a relative importance $\lambda \in \{\lambda_1,\ldots,\lambda_q\}$. It is a nine-point ordinal scale that provides differentiation in the degree of importance. The higher the importance value is, the more important the stakeholder is.

In brief, the purpose of release planning is to provide the most attractive features at the earliest releases to the most important stakeholders. For the purpose of this paper, $Value(s, f_i)$, $Urgency(s, f_i)$, and $Competitiveness(s, f_i)$ are the three attributes of a feature's attractiveness. Each feature can be prioritized from these three criteria with the value ranging from 0 to 9. These criteria are associated with the weights μ_1, μ_2, and μ_3, respectively.

The three prioritization criteria are the basis to formulate the objective of product release planning. The objective function $Utility(x)$ is defined as a linear combination of the priority votes of stakeholders related to these criteria:

$$Utility(x) = \sum_{k=1}^{K} (\xi_k \times \sum_{i:x(i)=k} \Pr iority(f_i))$$

where $Priority(f_i)$ is an aggregated priority of f_i defined as:

$$\Pr iority(f_i) = \sum_{s=s_1}^{S_q} (\lambda_s \times (\mu_1 \times Value(s, f_i) + \mu_2 \times Urgency(s, f_i) + \mu_3 \times Competitiveness(s, f_i)))$$

2.2 ReleasePlanner®

ReleasePlanner® [13] is a decision support system that aims at performing systematic product release planning based on computationally efficient optimization algorithms. Users are able to perform what-if analysis to pro-actively explore different scenarios defined by a sequence of inputs of the same project under investigation. In addition, the tool is capable of generating a set of diversified solution alternatives for each instance.

A series of studies on ReleasePlanner® revealed that its users tended to have higher confidence and trust on their manual solutions than the ones generated more efficiently by the tool [6]. The major reason is that the tool works in a black box manner and the rationale of solution generation is hard to understand by the users. This is even more complicated because the users usually investigate multiple scenarios with multiple solutions.

3 Concept Analysis

Concept analysis, firstly introduced in [17], is a theory of data analysis to identify conceptual structures among a set of data. It has been successfully applied to many fields [12], including in software engineering [11].

In this paper, concept analysis is investigated to address the three research questions presented in Section 1. Another two techniques, i.e. rough set analysis and dependency network analysis, have also been applied to explain release planning solutions by ReleasePlanner® [7]. However, they can only deal with the first two research questions and are not the focus of this paper. Detailed applications of these two techniques are available at [7].

3.1 Basic Terminology

Concept analysis investigates the relations R between a set of objects O, and a set of attributes A. The triple $C = (R, O, A)$ is called a formal context.

Def. 1 (Common Attributes and Common Objects): For any set of objects $\mathcal{O} \subseteq O$, the set of common attributes having the same attribute value is called common attributes related to $\mathcal{O}$ and is denoted by $ca(\mathcal{O}) = \{a \in A \mid \forall\, o \in \mathcal{O} : (o, a) \in R\}$. For a set of attributes $\mathcal{A} \subseteq A$, their common objects are $co(\mathcal{A}) = \{o \in \mathcal{O} \mid \forall\, a \in \mathcal{A} : (o, a) \in R\}$.

Def. 2 (Formal Concept): Each pair $c = (\mathcal{O}, \mathcal{A})$ with $\mathcal{O} = co(\mathcal{A})$ and $\mathcal{A} = ca(\mathcal{O})$ is called a formal concept. It demonstrates a pattern, i.e. relation, between $\mathcal{O}$ and $\mathcal{A}$.

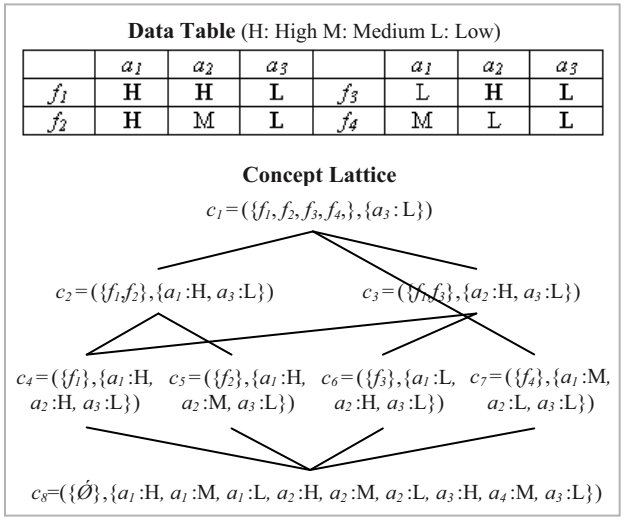
Def. 3 (Concept Lattice): All formal concepts for a given context C are called a complete concept lattice in which concepts can be partially ordered. If $c_1 = (\mathcal{O}_1, \mathcal{A}_1)$ and $c_2 = (\mathcal{O}_2, \mathcal{A}_2)$ are two concepts in the context C, a partial order $c_1 \leq c_2$ can be defined whenever $\mathcal{O}_1 \subseteq \mathcal{O}_2$ and $\mathcal{A}_1 \supseteq \mathcal{A}_2$.

Def. 4 (Greatest Lower Bound and Least Upper Bound): The greatest lower bound of c_1 and c_2 is the concept with objects $\mathcal{O}_1 \cap \mathcal{O}_2$ and attributes held by all objects in $\mathcal{O}_1 \cap \mathcal{O}_2$. The least upper bound of c_1 and c_2 is the concept with attributes $\mathcal{A}_1 \cap \mathcal{A}_2$ and objects which have all attributes in $\mathcal{A}_1 \cap \mathcal{A}_2$.

3.2 An Illustrate Example

Applied to planning product releases, O constitutes the set of features F to be assigned to different releases. The input to and output from product release planning using ReleasePlanner® form set A. Figure 1 shows a simple example of concept analysis in this domain. The upper part is a data table of feature set $F = \{f_1,\ldots,f_4\}$ defined with the attribute set $A = \{a_1,\ldots,a_3\}$. In this table, the value of each attribute for each feature can be H, M, or L.

These values represent different value ranges. The lower part of this figure shows the corresponding concept lattice with all the concepts $\{c_1,\ldots,c_8\}$ and their order relations. In this lattice, the values of the attributes in each concept are also highlighted. Among all the concepts, c_1 is the most general one and c_8 is the most specific one. Some of the order relations among the concepts are $c_2 \leq c_1$, $c_3 \leq c_1$, and $c_7 \leq c_1$. From this figure, we can also identify the least upper bound and greatest lower bound of a set of concepts. For example, c_1 and c_4 are the least upper bound and greatest lower bound of c_2 and c_3, respectively.



	a_1	a_2	a_3		a_1	a_2	a_3
f_1	H	H	L	f_3	L	H	L
f_2	H	M	L	f_4	M	L	L

Figure 1: Example concept analysis

4 Applying Concept Analysis to Explain Product Release Planning Results

4.1 Sample Project

To illustrate the application of concept analysis to explain results generated by ReleasePlanner®, we investigate on a sample project based on the data from a real life product release planning problem. As a summary, this project is defined with the following inputs:

- $F = 31$ features $\{f_1, \ldots, f_{31}\}$ to be assigned
- $K = 2$ releases
- $S = 18$ stakeholders $\{s_1,\ldots,s_{18}\}$ with weights $\{\lambda_1,\ldots, \lambda_{18}\}$
- Prioritization criteria *Urgency(s,f_i)*, *Value(s,f_i)*, and *Competitiveness(s,f_i)*
- $T = 3$ types of resources *{Res1*, *Res2*, and *Res3}*

The full details of this project setting can be referred to http://pages.cpsc.ucalgary.ca/~dug/ConceptAnalysis. This project setting and the results obtained from it are taken as the baseline scenario. From this baseline, the tool user also generates another 9 scenarios that the user thinks to be the most important (but not necessarily the complete) scenarios for investigation. Together these 10 scenarios are used for what-if analysis which is useful and important for product release planning, as discussed in Section 2.2. For all these scenarios, in total 35 solutions are generated by ReleasePlanner® for later analysis.

4.2 Data for Concept Analysis

With the above project settings, each feature f_i in each solution is associated with a data point used for concept analysis (see Table 1). The selection of the attributes is based on the experience of manual analysis of several product release planning projects [15]. The first six input attributes are relevant to stakeholder votes which are considered in the objective function for planning. In particular, *ConfUrgency(f_i)*, *ConfValue(f_i)*, and *ConfComp(f_i)* are the standard deviation between different stakeholders' votes for each feature f_i from the three criteria, respectively. They indicate the degree of disagreement among stakeholder opinions. The other six input attributes address resource utilization and criticality of features.

Table 1: Data defined for concept analysis

Input Attribute	Definition
AverageUrgency(f_i)	$AverageUrgency(f_i) = \frac{\sum_{S=S1}^{S18} \lambda_S \times Urgency(s, f_i)}{\sum_{S=S1}^{S18} \lambda_S}$
AverageValue(f_i)	(similar for *AverageValue(f_i)*, *AverageComp(f_i)*)
AverageComp(f_i)	
RelConfUrgency(f_i)	$\mathrm{Re}lConfUrgency(f_i) = \frac{ConfUrgency(f_i)}{AverageUrgency(f_i)}$
RelConfValue(f_i)	(similar for *RelConfValue(f_i)*, *RelConfComp(f_i)*)
RelConfComp(f_i)	
Res_tUtiRatio(f_i) (t = 1, 2, 3)	$\mathrm{Re}s_tUtiRatio(f_i) = \frac{r(f_i, \mathrm{Re}s_t)}{\sum_{i=1}^{31} r(f_i, \mathrm{Re}s_t)}$
Res_tCriticality(f_i) (t = 1, 2, 3)	$\mathrm{Re}s_tCriticality(f_i) = \mathrm{Re}s_tUtiRatio(f_i) - \frac{r(f_i, \mathrm{Re}s_t)}{\sum_{k=1}^{2} Cap(k, \mathrm{Re}s_t)}$
Output Attribute	**Definition**
Release(f_i)	*Release(f_i)*

Based on our previous experience on the analysis of these attributes, each attribute is discretized according to Table 2. The purpose of discretization is to scale attributes with continuous values to a nominal or ordinal scales.

Table 2: Discretization of the defined attributes

Input Attribute	Value Range	Discretization	
AverageUrgency(f_i)	A real number in [0, 9]	High	[6, 9]
AverageValue(f_i)		Medium	[4, 6)
AverageComp(f_i)		Low	[0, 4)
RelConfUrgency(f_i)	A real number in [0, 1]	High	[0.7, 1.0]
RelConfValue(f_i)		Medium	[0.4, 0.7)
RelConfComp(f_i)		Low	[0.0, 0.4)
Res_tUtiRatio(f_i)	A real number in [0, 1]	High	[0.10, 1.00]
		Medium	[0.05, 0.10)
		Low	[0.00, 0.05)
Res_tCriticality(f_i)	A real number in [-1, 1]	No	[0.00, 1.00]
		Low	(-0.01, 0.00)
		Medium	[-1.00, -0.01]
Output Attribute	**Value Range**	**Discretization**	
Release(f_i)	An integer in [1, 3]	Not necessary	

Based on the above definition and discretization, a table with 1085 data points (35 solutions with each containing 31 features) is obtained for later concept analysis. The complete table is available at the website provided earlier.

4.3 Concept Analysis of the Data

We used an open source library called Colibri/Java [3] to perform concept analysis. It builds a concept lattice which contains all patterns (concepts) for the data prepared in Section 4.2. We then implemented a tool to traverse the concept lattice to select only those patterns where the distribution of the output values significantly changed along the (subset) relations between these patterns. To test significance, we used Fisher Exact Value and Chi Square tests (significance level of p=0.01) [16]. Using our tool, two filtered lattices were generated that contain the patterns which affect the distribution of releases the most:

- Concept lattice #1 contains 64 patterns where the likelihood of assigning a feature f_i to release 1 is increased by at least 45%.
- Concept lattice #2 contains 1093 patterns where the likelihood of assigning a feature f_i to any release, i.e. 1, 2 or 3 (postponed), is increased or decreased by at least 30%.

The first lattice is essentially a part of the second one. The details of these lattices are available at the website given earlier and will be analyzed in depth in Section 5.

Table 3: Example record in the generated concept lattices

Context			*Res3Criticality(f_i)*_Low		
Var			*AverageComp(f_i)*_High		
Δ_R1	0.49	**Context_R1**	548	**Context+Var_R1**	118
Δ_R2	-0.3	**Context_R2**	339	**Context+Var_R2**	1
Δ_R3	-0.18	**Context_R3**	198	**Context+Var_R3**	0

Each filtered lattice consists of a number of patterns and transitions between these patterns in the form shown in Table 3. This table is read as, for all the 1085 cases in the dataset, the distribution of features f_i following the pattern of "*Res3Criticality(fi)* = Low" ("context" part) is 548 data points for release 1 ("context _R1"); 339 for release 2 ("context_R2"); and 198 for release 3 ("context_R3"). The pattern of "*Res3Criticality(fi*) = Low AND *Average Comp(fi*) = High" ("context" and "var") is supported by 118 data points for release 1 ("context+var_R1"); 1 for release 2 ("context+var_R2"); and 0 for release 3 ("context+var_R3"). The transition between these two patterns can be understood as a rule: adding "*Average Comp(fi*) = High" ("var") to the "context" part increases the likelihood of assigning a feature f_i to release 1 by 49% ("Δ_R1"), and decreases the likelihood to release 2 and 3 by 30% ("Δ_R2") and 18% ("Δ_R3"), respectively.

5 Analysis and Interpretation of Results

In this section, we analyze the two lattices generated in Section 4.3 from three perspectives: similarity of patterns, importance of input attributes, and sensitivity of outputs. The findings from these aspects contain new knowledge that reveals the underlying relationships between the inputs to ReleasePlanner® and its outputs, for the studied sample project. They can be used as explanations for this decision support system and its solutions.

5.1 Pattern Transitions and Data Similarity

Each generated concept lattice covers the most significant patterns discovered from the product release planning data used for concept analysis. These patterns are presented in the "context" part and of the different granularities, i.e. from the most general to the most specific. A general pattern can be transformed to more specific ones, and vice versa. By examining the generalization or specification relationships among these patterns, the transitions among the patterns become visible. In addition, the discovered patterns demonstrate the similarities shared among the data used for analysis. Data that are categorized under a same pattern are of the similarity as demonstrated by the pattern. For any two patterns that can be generalized to the same more general pattern, the two data sets supporting these patterns must be similar to each other in the way that is represented from the general pattern.

To illustrate the pattern transition and data similarity in this sample project, the concept lattice #1 is analyzed for simplicity. Any other lattices can be analyzed similarly.

Figure 2 shows the top four levels of patterns within this concept lattice and the transitions of these patterns. The complete transitions of all the patterns in this lattice can be referred to the website provided earlier. In this lattice, the most general pattern is #1, as shown in the very top of the figure. It can be specified to pattern #2, #3, and #4 at the second level. In this case, we say pattern #1 is the generalization of pattern #2, #3, and #4. On the other hand, pattern #2, #3, and #4 are the three specifications of pattern #1. Each of these three patterns can be further specified to other patterns until no more specific pattern can be found. For example, one of the most specific patterns is pattern #60. It follows the specification path of pattern #1→#3→#6→#41→#50→#60.

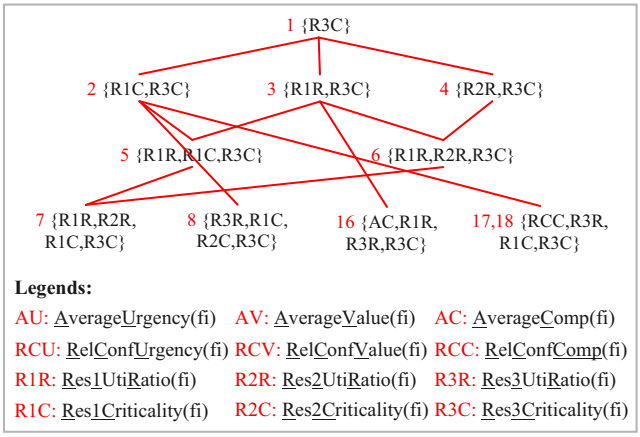


Figure 2: Transition of patterns (concept lattice #1)

Regarding the similarities shared among all the 1085 data used for the analysis, all these data are the same in terms of their values on the input R3C (*Res3Criticality(f_i)*), as illustrated through the most general pattern, i.e. pattern #1. More similarity is discovered from the data #1 to #715 and #869 to #930 because these data points also share the same value on the input attribute R1C (*Res1Criticality(f_i)*), besides on R3C. As a result, these data form a more specific pattern, i.e. pattern #2.

These results can be interpreted as a type of explanations on ReleasePlanner® solutions. If, in a solution, the release assignment of a feature is supported by general pattern(s) that are supported by a large number of data points, the users are more likely to accept such result. Otherwise, they might want to further improve the solution.

5.2 Importance Level of Inputs on Outputs

By examining all the found patterns ("context" part), we can identify each input attribute's importance level to the output attribute, in our case the release. The assumption is that the higher the number of the occurrence of an input in the patterns is, the more important this input is in determining the release value. However, an exception to this assumption is that this number cannot be as high as the total number of data used for analysis. The rational for this exception is given later.

For this purpose, we investigate the second concept lattice which provides more coverage than the first one on the patterns inherent in the data. Figure 3 summarizes the number of occurrence of each input in this concept lattice. *Res3Criticality(f_i)* appears to be the most important attribute. It is in all the patterns and has the same value. In other words, it has no influence at all on the distribution of release. *Res1Criticality(fi)* and *Res1UtiRatio(fi)* are important attributes which have different values. The least important ones are *AverageComp(fi)*, *AverageUrgency(fi)*, and *RelConfComp(fi)*. Other input attributes have medium level of importance.

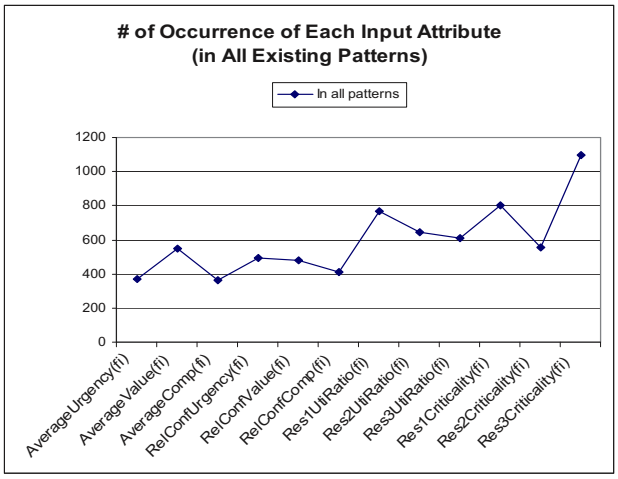


Figure 3: Importance level of each input on the output (concept lattice #2)

This part of the results provides the ReleasePlanner® users with the explanations by identifying a subset of all the defined inputs that play the most significant impacts on the tool when it generates solutions.

5.3 Sensitivity Level of Outputs to Inputs

The generated concept lattices also check if adding a new input attribute ("var" part) to the existing patterns ("context" part) would change the distribution of release. If the change is significant, this input is likely responsible for such change, i.e. the output is sensitive to this input. To observe the sensitivity of the output on each input, the second concept lattice is used for analysis again. In particular, we examine six types of how the "var" part may impact on the distribution of releases:

- R1/R2/R3 +0.30: increase by at least 30% in the distribution of assigning a feature f_i to release 1, 2, and 3, respectively
- R1/R2/R3 -0.30: decrease by at least 30% in the distribution of assigning a feature f_i to release 1, 2, and 3, respectively

For each input, we first count its number of occurrence in the "var" part of each record. For example, in the record in Table 3, the input attribute *AverageComp(f_i)* is in the "var" part with 118 cases supporting the impact of R1 +0.30. Therefore its number of occurrence in this record, for this type of impact, is 118. Then, by summing up such numbers for all the records of the same impact type, we obtain the total number of occurrence of this input. Figure 4 shows this number for each input based on the above calculation. We assume that the higher this number is, the more sensitive the output is to this input.

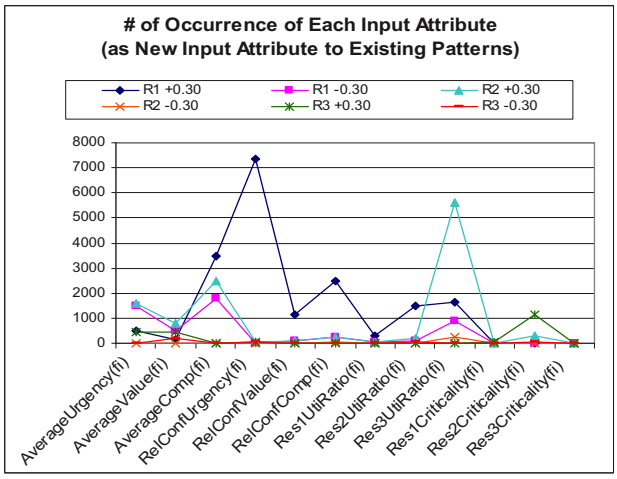


Figure 4: Sensitivity level of the output to each input (concept lattice #2)

From this figure, *RelConfUrgency(fi)*, *Res3UtiRatio(fi)* and *AverageComp(fi)* have the most significant impacts on the distribution of release. Specifically, release 1 is most sensitive to *RelConfUrgency(fi)* and *AverageComp(fi)* for at least 30% of increased distribution, and to *AverageComp(fi)* for at least 30% of

decreased distribution. Release 2 is most sensitive to *Res3UtiRatio(fi)* and *AverageComp(fi)* for at least 30% of increased distribution. But they usually have no impact as R2 -0.30 or R3 ±0.30. On the other hand, *Res1UtiRatio(fi)*, *Res1Criticality(fi)*, and *Res3Criticality(fi)* almost never contribute to any change by at least ±30% in any release. Although they occur the highest times in the patterns in Figure 3, their sensitivity levels are not as significant as at least ±30% and cannot be reflected in Figure 4. The other input attributes in general have medium level of sensitivity on the output attribute.

The above results explain some sensitivity aspects of the solutions generated by ReleasePlanner®. This kind of knowledge reveals the degree of impact from changing different input parameters. In case of uncertain data, the rule of thumb is that the more robust a solution, the higher chance of acceptance by the user.

6 Conclusions and Future Work

In this paper, a formal data analysis method called concept analysis is combined with statistical hypothesis testing to explain complex solutions recommended by ReleasePlanner®, a decision support system for product release planning. The results of our analysis of the data of individual release planning projects contain additional knowledge that is currently unavailable to the tool users. Specifically, such knowledge explains the underlying relationships inherent in the investigated data, in terms of the underlying patterns between the input and output data, as well as the importance and sensitivity levels of inputs on outputs. These explanations intend to achieve better understanding on the solutions of ReleasePlanner®, and therefore higher acceptance level from the user side. To demonstrate the application of concept analysis and statistical hypothesis testing, a sample product release planning project was investigated. Although the findings presented in this paper are specific to the sample project, the methodology of applying such analysis is generic (since it treats the decision support system as a black box) and can be applied to any other product release planning projects, or other software systems in which explaining complex solutions to users is necessary.

As a very important future work, empirical studies will be conducted with ReleasePlanner® users in order to justify the usefulness and effectiveness of the proposed method for explaining the tool's solutions. In addition, the results obtained from the concept analysis, as presented in this paper, only provides one type of explanations on ReleasePlanner® solutions and it is by no means complete. The explanations generated from this method are better to be used with other types of explanations (e.g. the ones discussed in [7]) that address the solutions from different aspects. Therefore we will also investigate on how these different types of explanations obtained from different techniques are complimentary to each other so that they can together provide the tool users with a more complete view of explanations on the tool and its solutions.

Acknowledgement

The authors would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Alberta Informatics Circle of Research Excellence (iCORE) for their financial support of this research. Many thanks are due to Daniel Götzmann and Christian Lindig who provided the Colibri/Java implementation.

References

[1] M. J. A. Berry, G. S. Linoff: Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management (2nd Edition), Wiley, 2004.

[2] P. Carlshamre: Release Planning in Market-Driven Software Product Development: Provoking an Understanding. Journal of Requirements Engineering, Vol. 7, 2002, pp. 139-151.

[3] Colibri/Java, available at http://code.google.com/p/ colibri-java/, last accessed February 2008.

[4] F. Davis, J. Kottemann: Determinants of Decision Rule Use in a Production Planning Task. Journal of Organizational Behavior and Human Decision Processes, Vol. 63, No. 2, 1995 pp. 145-157.

[5] M. Denne, J. Cleland-Huang: The Incremental Funding Method: Data Driven Software Development. IEEE Software, Vol. 21, No. 3, 2004, pp. 39-47.

[6] G. Du, J. McElroy, G. Ruhe: A Family of Empirical Studies to Compare Informal and Optimization-based Planning of Software Releases. Proceedings of the 5th International Symposium on Empirical Software Engineering, Rio de Janeiro, Brazil, 2006, pp. 212-221.

[7] G. Du, G. Ruhe: Comparison of Two Machine Learning Techniques for Explaining Results in Product Release Planning. Submitted to Journal of Information Sciences, Special Issue on Applications of Computational Intelligence and Machine Learning to Software Engineering, 2008, 36 pages.

[8] D. Greer: Decision Support for Planning Software Evolution with Risk Management. Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering, Banff, Canada, 2004, pp. 503-508.

[9] J. Karlsson, K. Ryan: A Cost-Value Approach for Prioritizing Requirements. IEEE Software, Vol. 14, No. 5, 1997, pp. 67-74.

[10] L. Lethola, M. Kauppinen, S. Kujala: Requirements Prioritization Challenges in Practice. Proceedings of the 4th International Conference on Product Focused Software Process Improvement, Vol. 3009, 2004, pp. 497-508.

[11] C. Lindig, G. Snelting: Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis. Proceedings of the 19th International Conference on Software Engineering, Boston, USA, 1997, pp. 349-359.

[12] U. Priss: Formal Concept Analysis in Information Science. Annual Review of Information Science and Technology, Vol. 40, 2006, pp. 521-543.

[13] ReleasePlanner®, available at www.releaseplanner.com, last accessed April 2008.

[14] H. W. J. Rittel, M. M. Webber.: Dilemmas in a General Theory of Planning. Policy Sciences, Vol. 4, 1973, pp. 155-169.

[15] G. Ruhe, A. Ngo-The: Hybrid Intelligence in Software Release Planning. Journal of Hybrid Intelligent Systems, Vol. 1, No. 2, 2004, pp. 99-110.

[16] L. Wasserman: All of Statistics: A Concise Course in Statistical Inference (2nd Edition), Springer, 2004.

[17] R. Wille: Restructuring Lattice Theory: an Approach based on Hierarchies of Concepts. In: Ordered Sets (Ed. I. Rival), Reidel, Dordecht-Boston, 1982, pp. 445-470.

Weighted Static Code Attributes for Software Defect Prediction

Burak Turhan, Ayse Bener
Dept. of Computer Engineering
Bogazici University
34342 Bebek, Istanbul, Turkey
{turhanb,bener}@boun.edu.tr

***Abstract*—It has been recently shown that defect predictors built on the combination of log-filtering, InfoGain attribute selection and Naive Bayes learner, outperform rule based learners. Naive Bayes is a well known statistical technique that assumes the 'independence' and 'equal importance' of attributes, which are not true in many problems. This paper addresses the 'equal importance' of attributes assumption of Naive Bayes. We show that with simple heuristics, relevant weights can be assigned to attributes according to their importance which improves defect prediction performance. Furthermore, our proposed heuristics have linear time computational complexities whereas choosing the optimal subset of attributes requires an exhaustive search in the attribute space. We compare the weighted Naive Bayes and the standard Naive Bayes predictors' performances on publicly available datasets both from Nasa and various small and medium enterprises (SMEs) in Turkey. Our results indicate that assigning weights to static code attributes may increase the prediction performance significantly, while removing the need for feature subset selection.**

***Index Terms*—Metrics/Measurement, Complexity measures, Methods for SQA and V&V**

I. Introduction

Quality of software is often measured by the number of defects in the final product. Minimizing the number of defects -maximizing software quality- requires a thorough testing of the software in question. On the other hand, testing phase requires approximately 50% of the whole project schedule [1], [2]. This means testing is the most expensive, time and resource consuming phase of the software development lifecycle. An effective test strategy should therefore consider minimizing the number of defects while using resources efficiently.

Defect prediction models are helpful tools for software testing. Accurate estimates of defective modules may yield decreases in testing times and project managers may benefit from defect predictors in terms of allocating the limited resources effectively [3].

Defect predictors based on linear regression, discriminant analysis, decision trees, neural networks, Bayesian networks and Naive Bayes classification have been analysed in previous research [4], [5], [6], [7], [8], [9], [10], [11]. Among these, Naive Bayes is reported to achieve significantly better performances than the other methods [8]. Naive Bayes assumes the independence and equal importance of attributes despite the fact that these assumptions are not true in many cases. Nevertheless, Naive Bayes has a good reputation for its prediction accuracy [12].

The number of research for relaxing the assumptions of Naive Bayes has significantly increased in recent years. These research focused on modifications to break the conditional independence assumption and weighting attributes [13], [14], [15], [16], [17]. All studies reported results that are generally 'not worse' than the standard Naive Bayes, while preserving the simplicity of the model.

For the attributes that are used for constructing predictors, some research prefer ranking the attributes for subset selection [18], [8], and some use a ranking criteria for attribute weight assignment [16], [19]. In fact, subset selection corresponds to 'hard' weighting of attributes, i.e. assigning 0 or 1 for attribute weights.

This paper attempts to tackle the assumption of "equal importance of attributes" of Naive Bayes in defect prediction context. Attribute weighting has been explored to some extent for other problems such as cost estimation. Auer et. al. employs attribute weigthing for analogy based cost estimation [19]. However, they assign *random* weights to project features and search for the optimal weigths. Similarly, neural network models for defect prediction have inherent attribute weighting. However, neural networks are non-deterministic and complex models that require optimization of the network structure together with many model parameters. Thus they require relatively large number of data samples for building a predictor. In practice, usually a limited amount of data is available. Further, the weights of the neural network model can not be easily interpreted especially in complex networks.

Furthermore, previous studies employ *feature subset selection* as a filtering step before learning a model. In brief, they label the available static code attributes as 'useful' or 'non-useful' based on some threshold determined by various attribute selection/ ranking methods. This approach, subsetting, has three disadvantages:

1. A threshold value should be set carefully either with extensive experimentation or manually.
2. An attribute labelled as useful is employed as if its degree of usefulness is the same as the rest of the useful attributes.

This research is supported in part by Bogazici University research fund under grant number BAP-06HA104.

3. An attribute labelled as non-useful:
   (a) may contain useful information, but is discarded by the threshold value.
   (b) may be useful in combination with other attributes, but the attribute selection/ ranking method discarded it [20].

In this paper we aim at removing these disadvantages introduced by feature subset selection. For this purpose, we propose attribute weighting along with several heuristics for determining the degree of importance of static code attributes. Recent research investigates integrating feature selection into generic classification models [21]. While not generic, our approach also integrates feature selection into Naive Bayes [17], [16].

Considering defect prediction, we claim that all static code attributes do not have equal effect on defect prediction and they should be treated accordingly. Our goal is to develop a methodology that permits the use of static code attributes in terms of their relevance to defect prediction. Menzies et.al. state that: *"how the attributes are used to build predictors is much more important than which particular attributes are used"* [8]. We also focus on *how* rather than which.

We reproduce the experiments on NASA datasets by Menzies et.al in order to construct a baseline for comparison. Furthermore, we include datasets collected from SME's, in order to justifiy the generalization of our results. We aim at combining the best practices of the above mentioned studies for constructing robust and accurate defect predictors. We use a weighted Naive Bayes classifier and construct heuristics for accurate attribute weight assignment. We propose to treat each attribute based on their estimated importance and we search for empirical evidence for the validity of our approach. Our results indicate that assigning weights to static code attributes may increase the prediction performance significantly, while removing the need for feature subset selection. Although the complexity of the proposed model is increased, we observe more stable results.

II. Methods

A. Weighted Naive Bayes (WNB)

Standard Naive Bayes derivation can be obtained by placing a special form of multivariate normal distribution, as the likelihood estimate in the Bayes theorem. By special form we mean that the off-diagonal elements of the covariance matrix estimate are assumed to be zero, i.e. the attributes are independent. In this case the multivariate distribution can be written as the sum of univariate normal distributions of each attribute (See Equation 1). In a classification problem we compute the posterior probabilities $P(C_i|x)$ for each class and choose the one with the highest posterior. In general the logarithms are used for computational convenience.

$$P(C_i|x) = -\frac{1}{2}\sum_{j=1}^{d}\left(\frac{x_j^t - m_{ij}}{s_j}\right)^2 + \log(\hat{P}(C_i)) \quad (1)$$

While assuming the independence of attributes, a weighting term can be introduced to reflect the relative importances of attributes. Then Weighted Naive Bayes can be written as in Equation 2 [16], [17].

$$P(C_i|x) = -\frac{1}{2}\sum_{j=1}^{d} w_j\left(\frac{x_j^t - m_{ij}}{s_j}\right)^2 + \log(\hat{P}(C_i)) \quad (2)$$

Now that we have introduced another parameter, we should find a way of estimating it accurately. For this purpose we propose 8 heuristics, which are explained in the next section.

B. Attribute Weight Estimation

We propose 8 heuristics mostly derived from attribute ranking techniques in order to estimate weights for the static code attributes. In all heuristics we compute the rank values for the attributes and then derive weights by normalizing over the sum of all rank values (See Equation 3). Thus, all weights are scaled to lie in the [0, 1] interval. A complete list of heuristics used in this research is given in Table I.

$$w_j = \frac{Rank(j)}{\sum_i Rank(i)} \quad (3)$$

First heuristics is based on the Principal Component Analysis (PCA), which projects the data points onto orthogonal principal axes such that the variance in each axis is maximized. We do not directly use PCA for dimensionality reduction. Rather, we claim that attributes with higher contributions for determining principal components should have higher weights in the prediction method. In our proposed heuristic, we use k eigenvalue and eigenvector pairs that correspond to the 95% of the proportion of the variance explained. Eigenvalues are written as $\lambda_1, \lambda_2, .., \lambda_k$ and eigenvectors are written as e_{id} where $i = 1..k, d = 1..D$ and D is the number of attributes. Then the weight of attribute d is estimated as a weighted sum of the corresponding eigenvector elements as given in Equation 4.

$$w_d = \frac{\sum \lambda_i e_{id}}{\sum \lambda_i} \quad (4)$$

Among proposed heuristics, GainRatio and InfoGain are mainly used in decision tree construction to determine the attributes that best splits the data [22]. Zhang and Sheng use the GainRatio heuristic for attribute weight assignment [16]. InfoGain is used in other studies for subset selection by ranking attributes [18], [8]. Our goal is to convert these ranking estimates into attribute weights. For this purpose we also evaluate OddsRatio, LogProb, ExpProb, CrossEntropy and Kullback-Leibler (KL) Divergence.

In defect prediction context, these heuristics correspond to the following: Given an attribute A,

- KL measures the similarity between the distributions of defective and nondefective modules. The more different the distributions, the higher the weight attribute A has.

TABLE I
List of heuristics used in this study.

Heuristic	Equation
PCA	See text
Information Gain	$IG(x,A) = Entropy(x) - \sum_{a \in A} \frac{\lvert x=a \rvert}{\lvert x \rvert} Entropy(x=a)$
Gain Ratio	$GR(x,A) = \frac{IG(x,A)}{SplitInfo(x,A)}$, where $SplitInfo(x,A) = -\sum_{a \in A} \frac{\lvert x=a \rvert}{\lvert x \rvert} \log \frac{\lvert x=a \rvert}{\lvert x \rvert}$
KL Divergence	$D_{KL}(x,A) = \sum_{a \in A} p(x=a\lvert pos) \log(x=a\lvert neg)$
Odds Ratio	$OR(x,A) = \log \sum_{a \in A} \frac{p(x=a\lvert pos)(1-p(x=a\lvert neg))}{(1-p(x=a\lvert pos))p(x=a\lvert neg)}$
Log Probability	$LP(x,A) = \log \sum_{a \in A} \frac{p(x=a\lvert pos)}{p(x=a\lvert neg)}$
Exponential Probability	$EP(x,A) = \exp\left(\sum_{a \in A} p(x=a\lvert pos) - p(x=a\lvert neg)\right)$
Cross Entropy	$CE(x,A) = \sum_{a \in A} p(x=a\lvert pos) \log p(x=a\lvert neg)$

TABLE II
Datasets

Name	# Attributes	# Modules	DefectRate (%)
CM1	38	505	9
PC1	38	1107	6
PC2	38	5589	0.6
PC3	38	1563	10
PC4	38	1458	12
KC3	38	458	9
KC4	38	125	39
MW1	38	403	9
SQ1	36	113	23
SQ2	26	286	5
SQ3	29	36	22
SQ4	26	1360	6
SQ5	26	1925	3

- OddsRatio measures wheter defective modules are more likely to occur than the nondefective modules.
- LogProb is the logarithm of the ratio of probability of a module being defective over probability of a module being nondefective.
- ExpProb is the exponentiation of the difference of probability of a module being defective and probability of a module being nondefective.
- CrossEntropy is the average number of bits needed to differentiate between the defective and nondefective module distributions.

Assigning weights with these heuristics takes linear time. On the other hand, ranking the attributes with these methods and then searching for an optimal subset requires both an exhaustive search in the attribute space and the evaluation of performance with each candidate subset. We expect to observe that the attributes that are discarded by the subset selection methods would have relatively small weights than the selected attributes.

III. Datasets

We have evaluated 13 datasets from different project domains. Among these, 8 public datasets are obtained from NASA MDP Repository [23]. These datasets are accepted to reflect the common industrial software engineering practice. However, in order not to restrict the experiments to datasets from a single company, we have also formed 5 more datasets (i.e. SQ datasets). These are collected from SME's in Turkey, which operate in different domains. These entities do not have any certification yet for their software processes, but some of them are in the process of certification and all are among the leading companies in their corresponding domains.

The projects in the SQ datasets include: embedded software from a leading whitegoods manufacturer, software for archieving media broadcasts for government audit, application software from a national science council, software for financial applications and telecommunication software.

Sample sizes of the projects vary from 36 to 5589 modules, which enables experiments in a broad spectrum of project sizes. Each NASA dataset has 38 attributes static code attributes (See [23], [8]). SQ datasets include 26 to 36 of the static code attributes available in NASA datasets. In both datasets modules with error counts greater than zero are assumed to be defective.

IV. Performance Measures

We have used probability of detection (pd) and probability of false alarm (pf) as the performance measures [8]. pd is a measure of accuracy for correctly detecting the defective modules. Therefore, higher pd's are desired. pf is a measure for false alarms and it is an error measure for incorrectly detecting the nondefective modules. pf is desired to have low values. Since we need to optimize two parameters, pd and pf, a third performance measure called balance is used to choose the optimal (pd, pf) pairs. balance is defined as the normalized Euclidean distance from the desired point (0,1) to (pd, pf) in a ROC curve [8].

Zhang and Zhang argue that using (pd,pf) performance mesures in imbalanced classification problems is not practical due to low precisions [24]. On the contrary, Menzies et.al argues that precision has an unstable nature and it can be misleading to determine the better predictor [25]. They also give examples of low precision predictors that are in use in SE industry. We also think that predictors with high pd rate can be practical even when they have high pf rate. Especially in mission critical and safety critical systems, detecting defects accurately with the cost of many false alarms is affordable [25], [26], [27]. Furthermore, we

believe that this strategy is still more efficient than using exhaustive testing. Thus, we argue that it would not defeat the purpose of defect prediction as Zhang and Zhang claims [24].

In addition, we would like to point out that balance performance measure should be used carefully for determining the best among a set of predictors. Since it is a distance measure, i.e. the distance of (pd,pf) to the optimal point (0,1), a specific balance value defines a quarter-circle on the ROC graph with radius of (1 - bal) and with the origin (0,1). So, predictors with different (pd,pf) values can have the same balance value. This does not necessarily show that all predictors with the same balance value have the same practical usage. As mentioned above, domain specific requirements may lead us to choose a predictor with a high pd rank although it may also have a high pf rank.

V. Experiment Design

We have compared the best defect predictor reported so far (log-filter, InfoGain attribute selection, standard Naive Bayes [8]) with the Weighted Naive Bayes classifiers constructed by our proposed heuristics on NASA datasets. Then we have evaluated the weighted Naive Bayes method on SQ datasets to show the applicability of the model in a wide range of company profiles. The experimental design follows the framework suggested as a baseline by Menzies et.al. [8]. We have also reproduced these experiments on NASA datasets for comparison purposes.

We have applied log-filtering on the datasets before we trained the predictor [8]. Then, we have used 10-fold cross-validation in all experiments. That is, datasets are divided into 10 bins, 9 bins are used for training and 1 bin is used for testing. Repeating these 10 folds ensures that each bin is used for training and testing while minimizing the sampling bias. Each holdout experiment is also repeated 10 times and in each repetition the datasets are randomized to overcome any ordering effect and to achieve reliable statistics. Overall, we have performed 10x10=100 experiments per heuristic for each dataset and our reported results are the means of these 100 experiments for each dataset. We have applied t-test with $\alpha = 0.05$ in order to determine the statistical significance of mean results. Since t-test assumes normal distribution, analysis can be misleading in case of skewness. In order to detect any skewness in the distribution of the results, we also include box-plots. All implementations are done in MATLAB environment.

VI. Results

(pd, pf, bal) results of 100 experiments for NASA and SQ datasets are plotted in Figure 1 and Figure 2 repectively. From these figures, we observe that Infogain(IG), GainRatio(GR) heuristics and standard Naive Bayes with log-filtering(LNB) outperforms other heuristics. These three methods show statistically significant performances than others in all datasets. Thus, we only tabulate these three methods' mean (pd,pf,bal) values in Table III and Table IV.

In NASA datasets, overall evaluation yields 5, 6 and

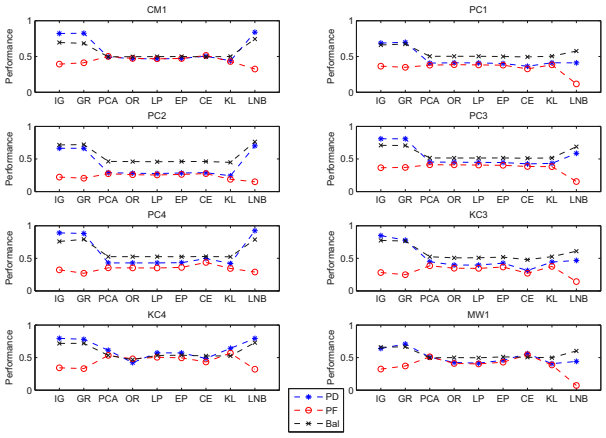


Fig. 1. Performance measures (pd, pf, bal) for Nasa datasets

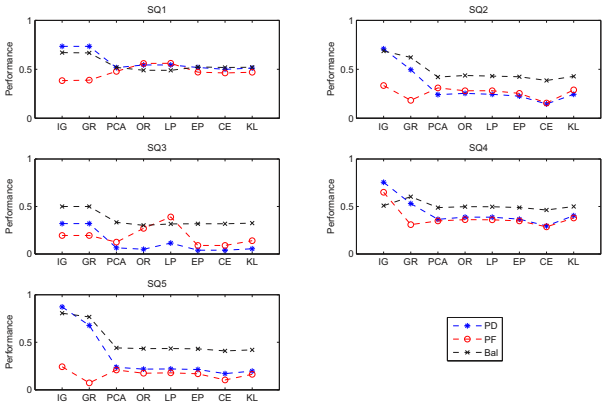


Fig. 2. Performance measures (pd, pf, bal) for SQ datasets

4 wins for InfoGain, GainRatio heuristics and LNB respetively. These results indicate that our proposed approach yields comparable and in some cases better results than the ones reported on these datasets so far. InfoGain and GainRatio heuristics achieve higher pd and pf values compared to LNB. We argue that the projects that require high reliability should have higher pd values. Since these datasets have this requirement, InfoGain and GainRatio based heuristics may be preferred over LNB.

In SQ datasets we observe a decrease in the average of all performance metrics. This stems from the bad predictions in SQ3 dataset. The results for SQ3 are (pd, pf, bal) = (32, 20, 50). Excluding SQ3 from the average enhances the results. On the other hand, SQ3 dataset includes only 36 modules whereas the number of attributes is 30. SQ3 results suggest that defect prediction becomes a harder task and it lacks the desired performance in small projects. However, this should be verified by testing on additional small projects.

Figure 3 shows the boxplots of 100 balance results for IG, GR and LNB. The leftmost and the rightmost lines in the boxes correspond to the 25% and 75% quartiles and the line in the middle of the box is the median. The notches around

TABLE III
RESULTS FOR NASA

Data	IG+WNB (%)			GR+WNB (%)			LNB (%)		
	pd	pf	bal	pd	pf	bal	pd	pf	bal
CM1	82	39	69	82	41	68	83	32	74
PC1	69	36	66	69	35	67	41	12	57
PC2	66	22	72	66	20	72	70	15	76
PC3	81	37	71	81	37	71	59	15	69
PC4	89	32	76	88	27	79	92	29	78
KC3	85	28	77	78	25	76	47	14	61
KC4	80	34	72	78	33	72	79	32	73
MW1	64	32	66	71	37	67	44	07	60
Avg:	77	33	71	77	32	72	64	20	69

TABLE IV
RESULTS FOR SQ

Data	IG+WNB (%)			GR+WNB (%)		
	pd	pf	bal	pd	pf	bal
SQ1	73	38	67	73	38	67
SQ2	71	33	69	50	18	62
SQ3	32	20	50	32	20	50
SQ4	75	65	51	53	31	60
SQ5	87	24	81	68	7	77
Avg:	68	36	64	55	23	63
Avg (-SQ3):	77	40	67	61	24	67

the median correspond to the 95% confidence interval of the median. We observe that the statistical signifinace of the means also apply for the medians of the three methods in most cases. We should also note that it is a sign of skewness if the medians are not centered between 25% and 75% quartiles. In most datasets, we observe such skewness. The dashed lines in the boxplots indicate 1.5 times of the interquartile range, i.e. the distance between the 25% and 75% quartiles. Data points outside these lines are considered as outliers. CM1, PC4, KC3 and MW1 datasets have relatively large number of outliers in this sense. Another observation is that the weighting results are more stable than standard Naive Bayes. Figure 3 shows that, in all datasets, the spread of balance values are less than or equal to that of Naive Bayes.

Figure 4 shows the same plot for SQ datasets. It is clearly seen that the behavior on SQ3 dataset is different from the others and the results are highly skewed due to the small number of data samples in this dataset.

VII. CONCLUSION AND FUTURE WORK

This paper presented an application of defect prediction built on weighted static code attributes. We have used several heuristics in order to estimate the weights of attributes based on their relative importance. These heuristics have linear computational times. We have evaluated our approach on Weighted Naive Bayes predictor, which is an extension of standard Naive Bayes. Our major contri-

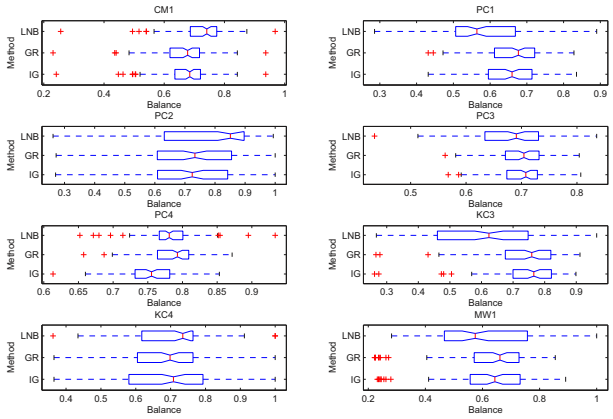


Fig. 3. Boxplots for Nasa datasets.

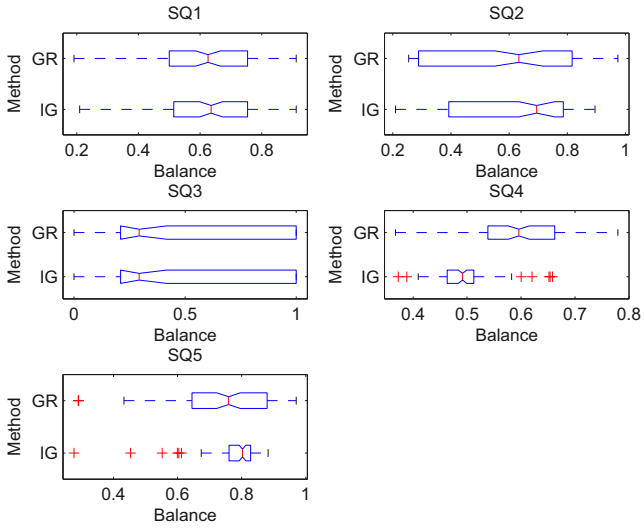


Fig. 4. Boxplots for SQ datasets.

bution in this research is to introduce a novel "weighted attributes approach" for defect prediction.

Condidering our motivation about the disadvantages of subsetting, we have eliminated the need for a threshold value, thus the most time consuming 'search for the optimal' step is avoided. All availabe attributes are used according to their estimated importances based on their discriminative power for defect prediction i.e. less important attributes are used with lesser weights. While some attributes seem to have greater importance than others for defect prediction, we believe in the notion that there is not a magical set of attributes to achieve the best performance. Moreover, automatic collection of static code attributes does not cost much. Therefore, we encourage to collect as many attributes as possible so that the weigthed scheme choses how they will be used. It would then be better to use all available attributes rather than explicitly throwing away a portion of collected data with subsetting.

Our proposed approach performs at least equaivalent and in some cases better than the currently best defect

predictor [8]. Although more parameters are introduced by the weights, the predictions are more stable. Finally, it has better time complexity and we generalize the results on both NASA datasets and datasets collected from various SME's.

Though some research stated against using static code attributes [28], [29], our results confirms the foundings of Menzies et. al.[8]. This shows the applicability of defect predictors based on static code attributes in a wide range of companies and projects.

We should note that our datasets include an extreme case (i.e. SQ3) with too few modules. We can comment that exceptionally small projects has highly skewed predictions with large variances. This is because there are not enough data samples to learn a stable theory. A future direction should be to focus on determining the necessary number of data samples for stable defect predictors.

From a software practitioner's point of view, these results are useful for detecting defects before proceeding to the test phase. In this sense, test resources can be managed more efficiently. Additionally, many companies in the software market develop their standards or make use of the best practices from industry, to determine the thresholds for static code attributes in order to guide developers during implementation. Our results indicate that the impact of changes in static code attributes to the defect rate varies for different attributes. Weights of the model can be interpreted as the attributes' contribution to the defectiveness of the modules and can be considered in code reviews.

One research direction is to examine the other assumption of Naive Bayes, which is the 'independence' of attributes assumption. We have ongoing research to address this problem by introducing multivariate distributions to defect predictors in order to model correlations between attributes.

Another future work is to measure attributes of module complexities by taking the software structure into account. Current static code attributes measure the complexites of modules independently, whereas these modules are not independent of each other. We think that attributes incorporating the communications among modules would lead better prediction performances.

References

[1] Mary Jean Harrold, "Testing: a roadmap," in *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, New York, NY, USA, 2000, pp. 61–72, ACM.

[2] Luay Ho Tahat, Atef Bader, Boris Vaysburg, and Bogdan Korel, "Requirement-based automated black-box test generation," in *COMPSAC '01: Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development*, Washington, DC, USA, 2001, pp. 489–495, IEEE Computer Society.

[3] Qinbao Song, Martin Shepperd, Michelle Cartwright, and Carolyn Mair, "Software defect association mining and defect correction effort prediction," *IEEE Trans. Softw. Eng.*, vol. 32, no. 2, pp. 69–82, 2006.

[4] J. Munson and Y. M. Khoshgoftaar, "Regression modelling of software quality: empirical investigation," *J. Electron. Mater.*, vol. 19, no. 6, pp. 106–114, 1990.

[5] John C. Munson and Taghi M. Khoshgoftaar, "The detection of fault-prone programs," *IEEE Trans. Softw. Eng.*, vol. 18, no. 5, pp. 423–433, 1992.

[6] Frank Padberg, Thomas Ragg, and Ralf Schoknecht, "Using machine learning for estimating the defect content after an inspection," *IEEE Trans. Softw. Eng.*, vol. 30, no. 1, pp. 17–28, 2004.

[7] Taghi M. Khoshgoftaar and Naeem Seliya, "Fault prediction modeling for software quality estimation: Comparing commonly used techniques," *Empirical Software Engineering*, vol. 8, no. 3, pp. 255–283, 2003.

[8] Tim Menzies, Jeremy Greenwald, and Art Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Software Eng.*, vol. 33, no. 1, pp. 2–13, 2007.

[9] nana Elena Pérez-Mi and Jean-Jacques Gras, "Improving fault prediction using bayesian networks for the development of embedded software applications: Research articles," *Softw. Test. Verif. Reliab.*, vol. 16, no. 3, pp. 157–174, 2006.

[10] Jean-Jacques Gras, "End-to-end defect modeling," *IEEE Softw.*, vol. 21, no. 5, pp. 98–100, 2004.

[11] Norman Fenton, Martin Neil, William Marsh, Peter Hearty, David Marquez, Paul Krause, and Rajat Mishra, "Predicting software defects in varying development lifecycles using bayesian nets," *Inf. Softw. Technol.*, vol. 49, no. 1, pp. 32–43, 2007.

[12] Pedro Domingos and Michael Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Mach. Learn.*, vol. 29, no. 2-3, pp. 103–130, 1997.

[13] David D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval," in *ECML '98: Proceedings of the 10th European Conference on Machine Learning*, London, UK, 1998, pp. 4–15, Springer-Verlag.

[14] Zijian Zheng and Geoffrey I. Webb, "Lazy learning of bayesian rules," *Mach. Learn.*, vol. 41, no. 1, pp. 53–84, 2000.

[15] M. Hall E. Frank and B. Pfahringer, "'locally weighted naive bayes," in *Proc. of the Uncertainty in Artificial Intelligence Conference*, 2003, pp. 249–256.

[16] Harry Zhang and Shengli Sheng, "Learning weighted naive bayes with accurate ranking," in *ICDM '04: Proceedings of the Fourth IEEE International Conference on Data Mining*, Washington, DC, USA, 2004, pp. 567–570, IEEE Computer Society.

[17] Mark Hall, "A decision tree-based attribute weighting filter for naive bayes," *Knowl.-Based Syst.*, vol. 20, no. 2, pp. 120–126, 2007.

[18] Dunja Mladenic and Marko Grobelnik, "Feature selection for unbalanced class distribution and naive bayes," in *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, San Francisco, CA, USA, 1999, pp. 258–267, Morgan Kaufmann Publishers Inc.

[19] Adam Trendowicz, Bernhard Graser, and Ernst Haunschmid, "Optimal project feature weights in analogy-based cost estimation: Improvement and limitations," *IEEE Trans. Softw. Eng.*, vol. 32, no. 2, pp. 83–92, 2006, Member-Martin Auer and Member-Stefan Biffl.

[20] Isabelle Guyon and André Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, 2003.

[21] Huan Liu and Lei Yu, "Toward integrating feature selection algorithms for classification and clustering," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, no. 4, pp. 491–502, 2005.

[22] Ross J. Quinlan, *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*, Morgan Kaufmann, January 1993.

[23] NASA, "WVU IV&V facility metrics data program," .

[24] Hongyu Zhang and Xiuzhen Zhang, "Comments on "data mining static code attributes to learn defect predictors"," *IEEE Trans. Softw. Eng.*, vol. 33, no. 9, pp. 635–637, 2007.

[25] Tim Menzies, Alex Dekhtyar, Justin Distefano, and Jeremy Greenwald, "Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'"," *IEEE Trans. Software Eng.*, vol. 33, no. 9, pp. 637–640, 2007.

[26] Burak Turhan and Ayse Bener, "Software defect prediction: Heuristics for weighted naive bayes," in *ICSOFT 2007*, 2007.

[27] Atac Deniz Oral and Ayse Bener, "Defect prediction for embedded software," in *ISCIS 2007*, 2007.

[28] Norman E. Fenton and Niclas Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Trans. Softw. Eng.*, vol. 26, no. 8, pp. 797–814, 2000.

[29] Martin J. Shepperd and Darrel C. Ince, "A critique of three metrics," *Journal of Systems and Software*, vol. 26, no. 3, pp. 197–210, 1994.

Predicting Software Project Size using Project Generated Information

Márcio de O. Barros
Postgraduate Information Systems Program – UNIRIO
Av. Pasteur 458, Urca – Rio de Janeiro, RJ – Brazil
marcio.barros@uniriotec.br

Abstract. In this paper we present a simulation based approach to predict the expected probability distribution that describes the size of a software project in a given period in the future. Since the simulation strongly depends on historical information, we propose the collection of such data from version control systems, which are well-known and widely used in the industry. We discuss the information collection process and the simulation model that is built and executed based on such data. Finally, we present a case study in which we apply the proposed approach to estimate the size of a large software project on a three month time frame, comparing simulation results with other estimation procedures.

Keywords: software estimation, software economics, measurement and empirical Software Engineering.

1 Introduction

Size prediction is a hard problem for software projects. Many are the external and internal forces that can influence software growth, and the combined effect of such forces may prove difficult to model. Nevertheless, estimating future size is paramount as an early warning sign of missing milestones and the need to negotiate adjustment to project schedule and cost baseline. By predicting the size of a software project in the near future, a manager can identify whether the development team is productive enough to attain the functionality necessary to meet the next milestones within the desired time frame. So, the manager can plan team allocation on maintaining the software assets that are already constructed and the new functions to be built.

Two perspectives are usually explored when researchers propose a software project prediction method: (a) modeling the project and its influence factors in great detail, yielding complex simulation models that can be executed under distinct scenarios to provide insight on the future of the software project [1] [2] [3]; or (b) selecting a small group of relevant influence factors, deriving analytical formulations that explain the attributes under estimation based on data from past projects, and applying these equations to estimate the project at hand [4] [5].

While the first perspective demands knowledge about the software project to build the simulation models, the second perspective requires a selection criteria by which past projects are chosen to provide data to estimate the new one. Usually, this selection criterion is based on similarity, a subjective and ambiguous term whose interpretation may vary according to the different concerns people present about the project. Also, external drivers (such as consumer's expected delivery date and budget restrictions) may influence the selection of "similar" projects, favoring those whose data yield estimations for the current project within a feasible zone. Finally, there may be issues regarding which data about a past project is available, how such projects were characterized, and how the information was stored. Diversity on data formats, technology used, programming language, development team, environment, and so on, may require adjustments on the available datasets in order to make them useful for the new project.

In this paper, we propose a simulation based approach to predict the expected probability distribution for the size of a software project in a given period, based on information collected from the project itself. Our fundamental assumption is that the most similar project to the one requiring estimation is the project itself. Thus, past information about the project is probably the best estimator for its future performance. Version control systems, like CVS [6] or SubVersion [7], commonly used in many industrial software projects, provide the required information. The approach is related to the second project prediction perspective, since relevant factors collected from past data are used to predict the future of the software project. A case study regarding the application of the approach is presented and its limitations are addressed.

The remaining of this paper is organized in five sections. Section 2 presents concepts and definitions that support the collection of information from version control systems to feed the simulation model. Section 3 presents the proposed simulation model and how results can be drawn from it. Section 4 presents a case study where the proposed approach is applied to a large software project. Section 5 presents related work. Section 6 addresses some limitations of the proposed approach and, finally, section 7 draw conclusions and directions for future work.

2 Collecting Information from Version Control

Configuration management tools have become well known and widely accepted by the software industry [8]. The availability of free source tools for version control, the growth of outsourcing and geographically distributed development, and the need for larger development teams drive the use of version control systems in medium and large software projects.

Version control systems store every version of relevant artifacts that compose a software project, thus maintaining a history of changes done throughout the software life-cycle. Version information, such as comments about the changes that were made, completion date and developer in charge, are attached to each version of an artifact. Version control systems are commonly used to allow navigation and retrieval of different versions of source code modules, but research in the field is evolving to allow requirements, design documents, project plans, and every other document to be managed by these systems.

Therefore, over the years the central repository of a version control system becomes a rich source of time-related information about a software project. After some development time, the repository is populated and the information it holds can be used to estimate relevant attributes for the project. So, instead of relying on "similar projects", our approach uses information from the project itself to support estimation.

Before describing the simulation process that yields the probability distribution for project size in a given period, we need to formalize the information structure under the control of a version control system. A version control system manages several software project repositories. We define a software project repository (SPR) as the finite set of files (F_i) that compose a software project:

$$SPR = \{F_i \mid 1 \leq i \leq N\}$$

Each file composing a software project repository is described by the directory in which it is located and an ordered set of reviews (R_i). Each review ($R_{i,j}$) is described by a unique identifier, the developer who committed the review to the central repository, the moment when the commit operation was executed, a comment describing the changes, and the resulting source code.

$$F_i = (\text{directory}, R_i)$$

$$R_i = \{R_{i,j} \mid 1 \leq j \leq M\} \ \therefore \ R_{i,j}.\text{time} < R_{i,k}.\text{time}, \ \forall \ j < k$$

$$R_{i,j} = (\text{id}, \text{developer}, \text{time}, \text{comment}, \text{code})$$

Based on such definitions, we define the *size* function, which represents the size of a given file review. The function maps a concept from the space defined by the reviews of a given file to the space of natural numbers, where size is defined.

$$\text{size}(R_{i,j}): R_i \Rightarrow N$$

We also define the *current* operator, which returns the last review committed in a given moment for a given file. It allows us to define the size function against time, measuring the size of the current review in that moment.

$$\text{current}(F_i, t) = R_{i,j}$$

$$\therefore R_{i,j}.\text{time} \leq t, \ R_{i,j} \in F_i$$

$$\therefore \text{not} \ \exists \ R_{i,k}: R_{i,j}.\text{time} < R_{i,k}.\text{time} \leq t, \ \forall \ 1 \leq k \leq M$$

$$\text{size}(F_i, t) = \text{size}(\text{current}(F_i, t))$$

Finally, we define the *diff* and *growth* operators. The first calculates the nominal size change observed from time t_1 to t_2, $t_1 < t_2$, while the second calculates the percentile size change in the period.

$$\text{diff}(F_i, t_1, t_2) = \text{size}(F_i, t_2) - \text{size}(F_i, t_1)$$

$$\text{growth}(F_i, t_1, t_2) = \text{diff}(F_i, t_1, t_2) / \text{size}(F_i, t_1)$$

The selection of a metric to act as the size function is not trivial. Lehman suggests counting the number of source files as a size measure for large software projects [9], while Godfrey and Tu [10] use lines of source code to measure size. Moreover, [11] has shown that the number of source code lines is high correlated to the number of source files in large software projects and, therefore, these metrics grow at roughly the same rate. We decided to use lines of code because most size estimation procedures are directly related to this measure, such as [4, 5].

Nevertheless, collecting information from version control systems is a time consuming activity [12] that must be automated to scale to large projects with many thousands of files. We have developed a framework, namely JCVS, which is able to collect information from version control repositories and store such data to XML files or database tables. These are easier to query for project specific information than the poorly structured textual files that compose the version control repository. In the proposed approach, our major interest is related to the number of lines of code in the files' reviews, but the framework is capable of retrieving the source code itself and all the attributes presented in the former formulations.

3 The Simulation Model

The historical growth data about the files composing a software project can be analyzed as a time series, that is, a set of values observed in successive time intervals. Converting time series to probability distributions (by computing the number of times each value is repeated in the series) is a practical procedure to highlight the series' descriptive characteristics, such as average value and variance. Moreover, by analyzing the correlations among time series, we can describe how they behave together, that is, how a change in one series is reflected in the other.

We propose the use of Monte Carlo simulation to estimate the size of a software project in a given period based on correlated probability distributions derived from time series collected from the project's version control system. Monte Carlo simulation is a sampling technique that estimates probability distributions for one or more results, given distributions for a set of parameters. The simulation process requires: (a) a finite set of parameters with known probability distributions; (b) a finite set of results, whose probability distributions will be estimated; and (c) a model that states how given parameter values yield a value for each result. Monte Carlo simulation consists in several cycles, each generating a value for each parameter (according to its probability distribution), calculating results according to the rules and relations prescribed in the model, and annotating these results. After thousands of such cycles, thousands of values are

annotated for each result and their probability distribution can be derived from these values. Monte Carlo simulation is popular because it allows parameters to be described by any type of probability distribution (such as the normal or beta distributions) and model development does not require specific knowledge on statistics or how to apply operations upon probability distributions.

In our proposed estimation method, the parameters are parts of the software project, being characterized by probability distributions estimated from growth time series extracted from version control systems; the single result is the observed growth of the project; and the model prescribes that the growth of the project is equal to the sum of the nominal growth of its parts divided by project size. So, the definition for both parameters and model depends on how the project is broken into parts. Two restrictions apply to this decomposition: (i) it must be possible to collect growth time series for each part from version control systems; and (ii) high correlated time series must be grouped and treated as a single part[1].

The first restriction allows two decomposition strategies: to take each file composing the project as a part or to take selected groups of files as a part. However, the second restriction eliminates the first strategy, since for large projects there would be thousands of composing files. In such a situation, there is a large probability that there will be any two files, F_i and F_j, so that changes to F_i will be coincident to changes to F_j over time. This results in high correlation among these time series, breaking the second restriction. Thus, the remaining option is to divide the project into file groups and treat each group as a part.

In the context of the proposed method, we define a component as a group of files that are logically related to each other in a project (for instance, they represent distinct design aspects for the same concept or process). The project must be divided into a set of complementary components (C_k). We suggest using of the project's directory structure as a starting point for the division, refining it according to the distribution of project features among the source code distribution units (for instance, packages in an UML model). The final set of components must be so that each and every file in the repository pertains to one and only one component.

$$SPR = \{C_k \mid 1 \le k \le Z\}$$

$$C_k = \{F_i \mid 1 \le i \le N\}$$

$$\therefore \forall F_i \in SPR \Rightarrow \exists\, C_k : F_i \in C_k,\ 1 \le i \le N,\ 1 \le k \le Z$$

$$\therefore \forall F_i \in SPR,\ F_i \in C_k \Rightarrow \text{not } \exists\, C_l : F_i \in C_l,\ 1 \le i \le N,\ 1 \le k, l \le Z;\ l \ne k$$

Given these definitions, we can define the size function and the *diff* and *growth* operators for components. The size function is defined below. The definition for the *diff* and *growth* operators is straightforward.

$$size(C_k, t) = \sum_{i=1}^{N} size(F_i, t) \qquad \therefore F_i \in C_k,\ 1 \le i \le N$$

After dividing the project into components, we collect a growth time series about each component from the version control system. These growth time series are calculated by applying the following process: (a) build a time series for the size of each project file, collecting these data directly from the version control system, as presented in section 2; (b) sum up the time series for the files that compose each component, yielding a time series for the size of each component; (c) apply the growth operator in a weekly basis over each component size time series, resulting in a weekly growth time series for each component.

The weekly basis was selected since we expect that our approach will be used in medium or large size projects, where development time is measured in months or years. Using a monthly basis would leave few data points to build the component's probability distribution, while a day seems too small a period to measure changes in components' sizes. Further investigation is required to determine if the weekly basis can be generalized to any project, independent of its size or any other characteristic.

Next, the growth time series for each component must be analyzed to identify and eliminate outlier points. Such outliers are common in software development, and they are discernible as ripples in the growth time series as files are reused into the component (upward ripple), files are transferred to another component (downward ripple), developers delay the commitment of large changes to the central repository (upward ripple), or third-party code is put under version control (upward ripple). We observed that such ripples are more common early in the project life-cycle or during the transition phase of a service acquisition (such as third-party development). Since they do not represent the expected behavior for the project, these outliers must be eliminated before simulation.

Each weekly growth time series is then converted to a probability distribution that describes the component in the simulation process. Correlations among these time series are calculated and organized in a symmetric matrix where each row and column represents a component and each cell conveys the correlation among the components in its row and column. After sampling a growth value for each component in a simulation cycle, such values are applied to the correlation matrix in order to represent the joint behavior of growth in components. Thus, we have a set of correlated parameter values.

The simulation process also depends on the estimation period, that is, the number of intervals after a given date on which project size will be estimated. As data collection is established in a weekly basis, the estimation period (Δ) is also expressed in weeks. Each simulation cycle samples correlated parameter values Δ times, yielding Δ random weekly growths for each component (G_t). The size of the component after the estimation period is given by:

[1] When using Monte Carlo sampling with correlated time series, their correlations are organized in a symmetric matrix that must be positive defined. This property cannot be observed if two or more series present high correlation.

$$size(C_k, T+\Delta) = size(C_k, T) \cdot \prod_{t=1}^{\Delta}(1+G_t)$$

After simulating each component size Δ weeks ahead, project size is estimated by summing up the sizes of its components. So, each simulation cycle yields an estimated project size and an estimated size for each component. By executing thousands of cycles, the proposed approach yields a probability distribution for project size and a probability distribution for each component's size.

4 Case Study

This section describes the application of the proposed approach to a large scale system. The system is used by financial institutions and financial departments of non-financial companies to identify, analyze, and manage investments in market, over-the-counter, and hedge assets. It is under development since early 2005, with an average team of ten developers, using object-oriented and software component technologies. The software is developed using an incremental process and is currently composed of about 500 KLOC distributed along 1,931 source code files.

In this case study we were interested in addressing the ability of the proposed approach to estimate project size in the future, comparing the estimated size to the observed growth. We had version control data for the project since February, 2005 to September, 2007. We designed the case study so that such data was separated into two samples: data from February, 2005 to June, 2007 was used to parameterize the simulation model, while data from July, 2007 to September, 2007 was used as a test sample, being compared to results achieved from simulation.

Based on an assessment of its directory structure and a meeting with the system architect, the software project was decomposed into fifteen components. Six of these are under development since the project started (February, 2005), five others were started a year later (February, 2006), and the last four components were started about a year and a half after development was commenced (November, 2006). Though some components had version control data available since February, 2005, we decided to collect information only for the period when all components were under development (that is, from November, 2006 to September, 2007). This allowed us 34 weeks of sample data and 12 weeks of test data. Also, this period would better reflect the actual productivity, since the development team was smaller in the early stages of the project life-cycle. Table 1 presents the size (in KLOC) of each component after the 34 weeks of sample data (first row) and after the 12 weeks of test data (second row).

Table 1 - Component sizes after the sample and test data periods

aux	sdbb	sass	sinf	srep	sdec	svar	scfr
21,7	45,0	51,7	53,7	6,1	23,8	28,0	20,7
24,9	48,1	51,8	54,5	14,0	28,2	33,7	21,3

idbb	iass	iinf	irep	idec	ivar	icfr
24,5	19,5	47,3	10,3	20,7	50,8	13,2
25,8	19,8	53,1	12,4	22,5	57,5	13,2

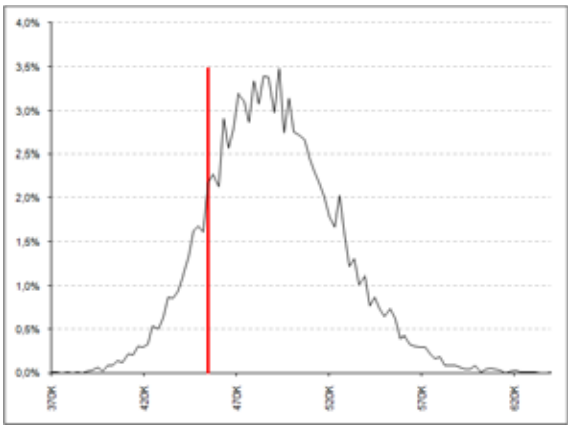
Next, we created the growth time series for the components, as presented in Section 3. By visually inspecting these time series, we observed some outliers and proceeded with an investigation with the system architect to understand whether these outliers were valid values that should be accounted for in the simulation or there were reasons for these large growth variations that should not repeat in the future. Therefore, we conducted a qualitative outlier elimination based on information gathered through interviews with the project architect, instead of a model based, quantitative outlier elimination procedure. This investigation revealed important issues about project components:

- The six components built from February, 2005 (*aux*, *sass*, *sinf*, *sdec*, *svar*, and *scfr*) were developed by reusing code from a previous version of the same system. So, their early development stages show high growth (thousands of KLOC added in a single week), which was not compatible with the effort dedicated to the components and could only be related to the inclusion of reused code. Therefore, data from the early development stages should be suppressed from the analysis. This was not an issue to the current case study (which used information from November, 2006 on), but was recorded as a lesson learned for field data collection;
- Two components (*srep* and *irep*) were developed by an external company, being integrated to the project's version control system later in their development process. So, the time series for these components present large ripples as new code is integrated monthly (due to a staged delivery schedule), instead of a smoother daily integration. Moreover, the maintenance of these components was transferred to the company within the test data period. So, the large growth ripples ceased and a moderate growth behavior was observed. This could not be projected by the components' time series, since they reflected a third-party development period. We decided to estimate project size without these components, thus eliminating their series from analysis;
- A component (*aux*) presented a "ladder style" time series: a week presenting high growth was followed by several weeks without development (that is, growth very close to zero). By enquiring the architect about this component, we discovered that it was composed by utilities classes, many of them reused from previous projects. The reuse of previous code justified the "ladder-effect" in the time series, as complete, quality code from other projects was injected into the version control repository. We decided to keep the component under analysis, eliminating a single very large ripple that made the component's size double in a single week.

After removing the outliers from the time series, we generated their probability distributions and calculated

their correlation matrix. The multivariate distribution was submitted to the simulation process, yielding future size distributions for each component. The distributions after 10,000 simulations are available at the URL http://www.uniriotec.br/~marcio.barros/seke2008/ index.html.

Figure 1 presents the probability distribution for project size 12 weeks after the sample data period, related statistical information, and a comparison to the observed size. The final project size (vertical line at 454.5 KLOC, excluding the *srep* and *irep* components) is within the boundaries of the probability distribution and close to the region determined by its first standard deviation from the average (469.7 to 510.1 KLOC).



Information	KLOC	Information	KLOC
Observed Size	454,5	Estimated Size	489,9
Std Deviation	20,2	Median	487,3
1st Quartile	465,5	3rd Quartile	511,8

Figure 1 - Probability distribution for future project size and related statistics

We compared this result with a projection of system size after 12 weeks using the growth of the whole system as a single time series. Over the 34 sample data points, the average growth for the whole system was about -0.91%, which yields an expected size of 376.8 KLOC after the 12 weeks of test data. Thus, the error between the simulation based estimation and the real system size after the test period was about 7%, while a projection of system size over the same period based on a single system growth time series would generate an error rate of about 17%.

We also compared simulation results to estimation based on exponential weighting the growth time series for the whole system, a usual estimation procedure in time series analysis. Exponential weighting uses a polynomial based on a factor (λ) by which observations are weighted. The polynomial is built so that recent observations have more weight than older observations. The estimated system growth (EG) is calculated as shown below.

$$EG = (G_t + \lambda.G_{t-1} + \ldots + \lambda^N.G_{t-N}) / (1 + \lambda + \ldots + \lambda^N)$$

$\therefore$ G_{t-K}: observed system growth on time t-K

By varying the lambda factor from 100% to 70%, we observed that estimation error for system size grows from 17% to 20%. Thus, correlations among component growth time series play a fundamental role to estimation, which is not captured by analyzing the single system growth path.

Analyzing the distribution for component sizes, we observed that *svar* presented higher growth than expected from past information, *idbb* presented lower growth, and *idec* presented less growth variations than observed in the time series. Questioning the architect about such results, we were informed that new features were recently added to the *svar* component (leading to higher growth than expected), and that sample data captured a phase in which new features were implemented in *idbb*, complementing features that were earlier implemented in the *sdbb* component (thus, test data represented an stabilization period for such component, presenting lower growth). Finally, the architect could not explain the large variability in the *idec* component in recent, test data.

Thus, analysis results provide indications that the proposed method may be useful to address project size in the future. Moreover, analysis indicates that it can elicit relevant questions about project component's evolution, useful to register project history and to predict future behavior for the software project.

5 Related Work

The most typical approaches for using version control data to provide insight upon a software project are based on visualization. Many approaches, such as the Seesoft tool [13], the Aspect Browser [14], and Augur [15], allow the visualization of dependencies among changes to a software project. Such tools usually summarize project information in line lengths and/or color codes, allowing thousands of data elements to be presented simultaneously on a screen. However, these tools are not yet able to draw conclusions from the data, leaving this task to the analyst.

Monte Carlo simulation has been used to draw the evolutionary path of specific uncertainties in software projects. Grey [16] and Hullet [17] present simulation methods to estimate project schedule and cost baseline. In [2], we see the use of Monte Carlo sampling to model uncertain results of a system dynamics model, according to probability distributions associated to its parameters.

To the best knowledge of the author, there is no attempt to use size information from version control systems to predict the future size of software projects. However, in a recent paper Kitchenham et al [18] addressed the benefits of using in-company estimation methods (that is, estimation models based on information from previous projects developed by the same company) instead of using cross-company methods. It was observed that, in some companies, in-company methods performed better than cross-company models, thus enforcing that in-project data may be useful for estimation.

6 Limitations of the Proposed Approach

Due to the nature of the information used to describe the parameters of the simulation process, the proposed

approach has some limitations. First, it is tightly related to the coding activity, since version control systems are usually used during such activity. However, we see growing interest on using such systems before coding, to store and evolve specifications, design diagrams, and other artifacts created during project development. As research in this direction unfolds, we probably will be able to use other artifacts to support estimation.

Second, the approach cannot estimate project size early in the life-cycle, since there must be "past information" upon the project. We suggest that analogy estimation, based on similar projects from the past, might be used to provide a first estimation for the project. This estimation can be refined as project information is made available in version control systems. The combination of analogy and project-based estimation strategies requires investigation and is a future perspective for our research.

We also believe that the usefulness of the proposed approach may vary according to the usage of different project life-cycle models. Project-based estimation is probably more useful when development is carried on using an incremental or spiral life-cycle model, in contrast to a waterfall life-cycle model, since the coding activity (and generation of version control information) starts earlier in the former models.

Other important aspect of the proposed method is that estimation is solely based on the production of source code lines. Though it may seem that many factors that influence the growth of a software project are overlooked, the method assumes that recent history will repeat itself in the near future. Thus, the same factors that affected recent production of source code lines will play their role again, affecting production in the estimation time horizon. So, we rely on a single relevant productivity indicator, assuming that it captures the influence of other factors.

7. Conclusions and Future Perspectives

This paper presented a simulation process to estimate the size of a software project in a given time horizon, according to information collected from the project's version control system. The project is divided into components, component size changes in the past are calculated from version control data, probability distributions are drawn to describe component growth dynamics, and Monte Carlo simulation is used to estimate the joint effect of these distributions upon project size.

Further investigation is being conducted to assess the usefulness of the approach under distinct development models. We intend to evaluate it under agile development, process-oriented organizations, and distinct life-cycle models. We also intend to provide better procedures for the component division and outlier elimination stages of the proposed approach. Moreover, since Monte Carlo simulation is computer intensive, we can study version control information on the perspective of time series models, such as auto-regressive (AR), moving average (MA) and integrated models (ARIMA).

Acknowledgments. The author would like to thank and acknowledge the support from the Brazilian Research Council (CNPq) and the Foundation for Scientific Development of the Rio de Janeiro State (FAPERJ).

References

1. Abdel-Hamid, T., Madnick, S.E. Software Project Dynamics: an Integrated Approach, Prentice-Hall Software Series, Englewood Cliffs, New Jersey (1991)
2. Barros, M.O., Werner, C.M.L., Travassos, G.H.: Supporting Risks in Software Project Management. Journal of Systems and Software 70 (1-2): pg 21-35 (2004)
3. Pfahl. D., Al-Emran, A., Ruhe G.: A System Dynamics Simulation Model for Analyzing the Stability of Software Release Plans. Software Process: Improvement and Practice, Volume 12: pg 475-490 (2007)
4. Boehm, B.W., Clark, B., Horowitz, E., Westland, J.C., Madachy, R.J., Selby, R.W.: Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. Annals of Software Engineering. 1: 57-94 (1995)
5. Putnam, L.H. A General Empirical Solution to the Macro Software Sizing and Estimating Problem. IEEE Transactions on Software Engineering, Vol. 4, 345 – 361 (1978)
6. Concurrent Versions System. http://www.nongnu.org/cvs/
7. Subversion. http://subversion.tigris.org/
8. Voinea, L., Lukkien, J., Telea, A..: Visual Assessment of Software Evolution, Science of Computer Programming, Volume 65, pg. 222–248 (2007)
9. Lehman, M.M., Ramil, J.F.: An Approach to a Theory of Software Evolution, International Workshop on Principles of Software Evolution, Vienna, pg 10-11 (2001)
10. Godfrey, M.W., Tu, Q., Evolution in Open Source Software: A Case Study, Proceedings of the International Conference on Software Maintenance, San Jose, EUA (2000)
11. Herraiz, I., Robles, G., González-Barahona, J., Capiluppi, A., Ramil, J.: Comparison between SLOCs and number of files as size metrics for software evolution analysis, Intl. Conference on Software Maintenance and Reengineering (CSMR'06), Bari, Italy (2006)
12. Voinea, L., Telea, A.: An Open Framework for CVS Repository, Querying, Analysis and Visualization, Intl. Workshop on Mining Software Repositories, Shangai (2006)
13. Eick, S.G., Steffen, J.L., Sumner, E.E.: Seesoft — a Tool for Visualizing Line Oriented Software Statistics, IEEE Transactions on Software Engineering 18 (1992) pg 957–968
14. Griswold, W.G., Yuan, J.J., Kato, Y.: Exploiting the Map Metaphor in a Tool for Software Evolution, Proceedings of the International Conference on Software Engineering, IEEE CS Press, Washington, USA, (2001), pp. 265–274
15. Froehlich, J., Dourish, P.: Unifying artifacts and activities in a visual tool for distributed software development teams, Proceedings of the International Conference on Software Engineering, IEEE CS Press, USA, pp. 387–396 (2004)
16. Grey, S., Practical Risk Assessment for Project Management. Wiley (1995), ISBN: 047193979X
17. Hullet, D., Schedule Risk Analysis Simplified. PM Network, July (1996), pp 23 – 30
18. Kitchenham, B., Mendes, E., Travassos, G.H., Cross- vs. Within-Company Cost Estimation Studies: A Systematic Review. IEEE Transactions on Software Engineering, Volume 33, Issue 5 (2007), pp 316 – 329

Supporting Reusable Component Selection with Use Case Gap-Based Development Effort Estimation

Xin Zhou, Bonnie Ray, Chenhua Feng
IBM China Research Lab, Beijing 100094, China
{zhouxin, bonnier, fengch}@cn.ibm.com

Abstract

Reuse-based development effort is an important factor to be considered when selecting appropriate reusable components. However, it's rarely considered very seriously in current practice, as current methods for estimation of reuse development effort rely heavily on personal experience and different developers may provide very diverse estimates. In this paper, we propose a method - AREA (Asset Reuse Econometric Analysis) that enables systematic evaluation of development effort for a new software based on consideration of alternative reusable components. The core components of the method are a process guiding the evaluation and an algorithm for calculating the effort based on use case gap analysis between to-be developed new software and selected reusable components. A proof-of-concept implementation of AREA is introduced and the feedback acquired through its pilot with a solution development team in IBM's Global Business Solutions Center (GBSC) is presented.

Keywords: software reuse, reusable component selection, development effort estimation, use case gap

1. Introduction

Software applications are playing an increasingly critical role in supporting daily business, and there is widespread need for high quality software applications to be delivered in very tight time schedule. Software reuse has been considered a promising way for improving the quality and productivity of software development via leveraging validated reusable software components [1][2].

Among the issues and problems associated with software reuse, the selection of suitable reusable components from numerous candidates is a critical one [3]. The challenge is to achieve appropriate balance among multiple concerns: functional requirements, non-functional requirements, technical compatibility, financial issues, and so on. An overall reusable component selection process and some general criteria for assessing the suitability of potential reusable components are presented in [4]. A strategy for managing risks in the components selection is proposed in [5]. Kontio et. al present their COTS (Commercial Off-The-Shelf) selection method – OTSO and its case studies in [6][7]. Also there are requirement engineering based methods for COTS evaluation and selection, as introduced in [8][9]. These existing works mainly focus on systematic selection process, selection criteria definition technique, functional and non-functional requirement based selection, and decision making techniques. However, they rarely consider the potential development effort for developing the new software, leveraging a given set of candidate reusable components. The lack of careful consideration of reuse-based development effort may cause inappropriate selection of reusable components, which increases the possibility of higher cost and longer development lifecycle and counteracts the benefit of reuse.

In this paper, we present AREA (Asset Reuse Econometric Analysis), a method that enables systematic evaluation of the development effort for each candidate reuse approach based on alternative reusable components selection. The core components of the method include a process guiding the evaluation and an algorithm for calculating the effort based on analyzing the use case gap between to-be developed new software and selected reusable components. A proof-of-concept implementation is built for AREA and is currently being piloted with solution development teams working at IBM's Global Business Solutions Center (GBSC), from whom preliminary feedback has been acquired.

The remainder of this paper is organized as follows. Section 2 describes the AREA process and algorithm. Section 3 introduces a web-based AREA support tool. We report the pilot of AREA in section 4. In section 5, we conclude and propose some future works.

2. AREA Method

In this section, we will first describe the conditions on which AREA is intended to be applied. Then, the overall AREA process and the core use case gap-based development effort calculation algorithm are presented. Finally, we will give an illustrative example for better understanding of the method.

2.1. Assumptions

In present reuse practice, various public or private reusable component repositories [10][11] are set up to accommodate reusable component information and software developers search and/or retrieve reusable components for reuse based software development. To apply the AREA method, we assume that for each reusable component registered in the repository, the following affiliated artifacts can be found: its use cases, its implementation work products, the traceability between use case and work products providing information on which work products realize which use cases.

2.2. The AREA Process

Figure 1 describes the overall AREA process. There are four major steps in the process: searching reusable components, generating reuse approaches, calculating development effort for each reuse approach, and comparing all the reuse approaches considering their development effort.

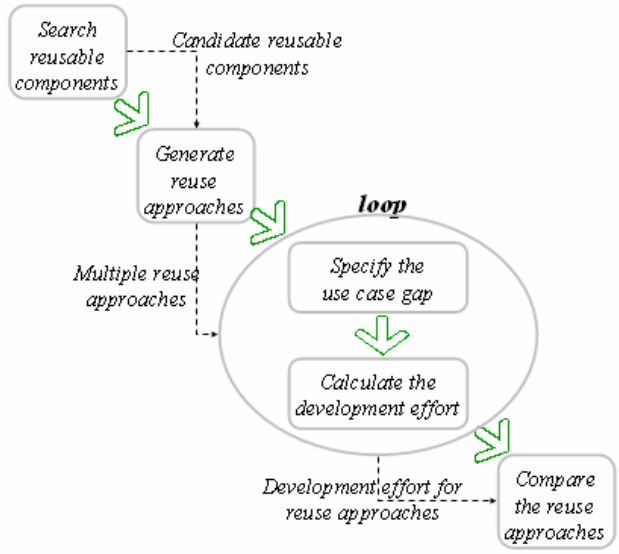


Figure 1: Overall AREA Process

First given the to-be developed new software, software developers search a reusable component repository for candidate reusable components that fully or partially realize the whole or partial requirement of the to-be developed software. For example, we might find three candidate reusable components for required function "xml parser written in java" and five candidate reusable components for requirement "logging".

In the second step, based on the search result, software developers can generate multiple reuse approaches. For example, in approach one, candidate reusable components a, b and c will be reused, while in approach two, candidate reusable components a, e, and f will be reused.

While generating the reuse approach, some factors are usually considered by software developers, including the to-be developed software's architecture design, the to-be developed software's technical constraints, and the interface compatibility of reusable components selected for this approach, etc. It's rare that a resulting reuse approach can exactly match the requirements of the to-be developed software. As a result, once we take one approach, further efforts are required to supplement new functions, to transform existing un-matched functions, and to remove unwanted functions, and the effort size depends on the degree of function added, transformed or deleted. In order to calculate the development effort, we need to clearly specify the gap between requirements of the to-be developed software and the existing functions of the individual components considered for reuse. As use case [12] has been widely adopted as an effective way to specify software functions, AREA recommends specifying the requirement gap with use case gap metrics.

Use Case Points (UCP) estimation method estimates software development effort based on external use cases specifying the software's functional requirements [13]. UCP method assumes that all use cases are to be developed from scratch. Use case points are assigned according to use case and actor's complexity and adjusted by a set of technical factors and environment factors. The final effort estimation is calculated by multiplying UCP by a statistical parameter PHperUCP (Person Hours per UCP). Experiments [14][15] reveals that this method is more accurate than expert estimation in industrial trials. Our work differs from this in that we estimate the development effort based on use case gap between to-be developed software and reusable components, instead of use case of the to-be developed software.

Table 1 provides a non-exhaustive list of the basic use case gap metrics.

Table 1: Basic Use Case Gap Metrics

Object	Metric Definition
Actor	(Weighted) Number of as-is actors
	(Weighted) Number of new actors
	(Weighted) Number of unwanted
	(Weighted) Number of to-be transformed actors
Event Flow	(Weighted) Number of as-is flows
	(Weighted) Number of new flows
	(Weighted) Number of unwanted flows
	(Weighted) Number of to-be transformed flows
Extension Point (EP)	(Weighted) Number of as-is EPs
	(Weighted) Number of new EPs
	(Weighted) Number of unwanted EPs
	(Weighted) Number of to-be

For a given use case of the to-be developed software, there are three scenarios when identifying reusable components. The first case is that it represents totally new functions and cannot find any similar reusable component use case. The second case is that there is one reusable component use case representing similar functions as it does. The third case is that there are multiple reusable component use cases which together provide similar functionality.

Based on the clearly specified use case gap, we will leverage the size and effort information of the reusable components selected in current reuse approach to calculate the potential development effort for the to-be developed new software taking this approach. Our ultimate purpose is not to provide a very precise estimation on the development effort for resource and schedule planning, but to provide software developers with an additional effort related dimension for reuse approach tradeoff. The effort calculation approach is detailed in the following section

2.3. The Effort Calculation Algorithm

For the convenience of algorithm presentation, we first define some notation as follows:

- The reusable components selected by current reuse approach are denoted as C_1, C_2, …, C_k
- The use cases of the to-be developed new software are denoted as T_UC_1, T_UC_2, …, T_UC_n
- The use cases of a reusable component C_i are denoted as R_UC_{i1}, R_UC_{i2}, …, R_UC_{im}
- The reuse relationship between to-be developed software use case and reusable component use case(s) is denoted as (T_UC_i, $Set\_of\_R\_UC_i$).
- The implementation work products related to a reusable component use case set Set_of_R_UCi are denoted as $WP_1\_UC_i$, $WP_2\_UC_i$, …

Considering the three scenarios about the similarity between T_UC and R_UC introduced in previous section, there may be none, one, or multiple reusable component use cases in $Set\_of\_R\_UC_i$. The development effort E for new software consists of the effort E_{new} for realizing those use cases without leveraging any reusable components and the effort E_{reuse} for realizing those use cases leveraging reusable components.

Formula 1:

$$E = E_{new} + E_{reuse}$$

Formula 2:

$E_{new} = \sum$ Effort of T_UC_i, where $|Set\_of\_R\_UC_i| = 0$;

Formula 3:

$E_{reuse} = \sum$ Effort of T_UC_i, where $|Set\_of\_R\_UC_i| > 0$;

For the effort of T_UC_i whose Set_of_R_UCi is empty, we still need software developers to manually input a development effort as there is no baseline for reference. Summing these effort values comes to E_{new}.

For the effort of T_UC_i whose Set_of_R_UCi is not empty, we will calculate it based on the historical development effort of the implementation work products it reuses. Suppose the gap between T_UC_i and $Set\_of\_R\_UC_i$ is denoted by P_{Ai}, P_{Ni}, P_{Ti}, and P_{Di}. More specifically, P_{Ai} denotes the percentage of $Set\_of\_R\_UC_i$'s actors/event flows/extension points that have the same counterpart from use case(s) in T_UC_i. P_{Ni} denotes the ratio of new actors/event flows/extensions points in T_UC_i to the total actors/event flows/extensions points in $Set\_of\_R\_UC_i$. P_{Ti} denotes the percentage of $Set\_of\_R\_UC_i$'s actor/event flow/extension point that has similar but not equal counterpart from use case(s) in T_UC_i. P_{Di} denotes the percentage of $Set\_of\_R\_UC_i$'s actor/event flow/extension point that has neither similar nor equal counterpart from use case(s) in T_UC_i.

The effort calculation for this kind of T_UC_i is defined by formula 4. Here, we make an assumption that reusing as-is from reusable components needs no effort.

Formula 4:

Effort of $T\_UC_i = (P_{Ti} * TF_i + P_{Ni} + P_{Di} * DF_i) * \sum$ Effort of $WP_j\_UC_i$, where $|Set\_of\_R\_UC_i| > 0$;

TF_i (Transformation Factor) is a number denoting the ratio of transformation difficulty to new development difficulty. For instance, a TF_i value 0.5 means that for use case T_UC_i, developing from scratch is twice as hard as transforming based on existing implementation work products. Similarly, DF_i (Deletion Factor) is a number to denote the ratio of deletion difficulty to new development difficulty.

2.4. An Illustrative Example

In this section, we will use an example to demonstrate the AREA process and calculation algorithm.

** Search Reusable Components*

Suppose the to-be developed software S has three use cases: T_UC_1, T_UC_2, T_UC_3. The candidate reusable components obtained from repository are C_1 (with one use case R_UC_{11}), C_2 (with two use cases R_UC_{21} and R_UC_{22}) , C_3 (with two use cases R_UC_{31} and R_UC_{32}) and C_4 (with one use case R_UC_{41}). The use case similarity relationship between the new software S and C_1, C_2, C_3, C_4 are shown in Table 2.

Table 2: Use Case Similarity Relationship

Use Case		*T_UC1*	*T_UC2*	*T_UC3*
C_1	R_UC_{11}	Yes	No	No
C_2	R_UC_{21}	No	Yes	No
	R_UC_{22}	No	No	Yes
C_3	R_UC_{31}	Yes	No	No
	R_UC_{32}		No	No
C_4	R_UC_{41}	No	Yes	No

Also, software developers can get the historical development efforts of implementation work products realizing C1, C2, and C3 from repository, as listed in Table 3.

Table 3: Historical Work Product Development Effort

Use Case		*Work Product*	*Development Effort*
C_1	R_UC_{11}	WP_{11}	20PHs (*)
C_2	R_UC_{21}	WP_{21}	45PHs
	R_UC_{22}	WP_{22}	30PHs
C_3	R_UC_{31}	WP_{31}	30PHs
	R_UC_{32}	WP_{32}	20PHs
C_4	R_UC_{41}	WP_{41}	25PHs

(* PHs means PersonHours)

** Generate Reuse Approaches*

Based on the preliminary analysis on above search results, software developers generate two alternative reuse approaches. For the first approach, C_1 and C_2 are reused. For the second approach, C_3 and C_4 are reused.

** Development Effort Calculation for Reuse Approach 1*

As in reuse approach 1, C_1 and C_2 will be reused by S, the software developers compare the use cases of S and those of C_1 and C_2. They get the values for use case gap metrics, as listed in table 4.

Table 4: Use Case Gap for Reuse Approach 1

	P_A	P_N	P_T	P_D
(T_UC_1, {R_UC_{11}})	60%	20%	20%	20%
(T_UC_2, {R_UC_{21}})	70%	30%	30%	0
(T_UC_3, {R_UC_{22}})	90%	10%	10%	0

Suppose the transformation factor $TF_1 = TF_2 = TF_3 = 0.6$ and deletion factor $DF_1 = DF_2 = DF_3 = 0.4$, then the calculation steps for reuse approach 1's development effort is:

(1) Effort of T_UC_1
= $(P_{T1} * TF_1 + P_{N1} + P_{D1} * DF_1) * \sum$ Effort of $WP_j\_UC_1$
= (20% * 0.6 + 20% + 20% * 0.4) * (20)
= 8 PHs

(2) Effort of T_UC_2
= $(P_{T2} * TF_2 + P_{N2} + P_{D2} * DF_2) * \sum$ Effort of $WP_j\_UC_2$
= (30% * 0.6 + 30% + 0 * 0.4) * (45)
= 21.6 PHs

(3) Effort of T_UC_3
= $(P_{T3} * TF_3 + P_{N3} + P_{D3} * DF_3) * \sum$ Effort of $WP_j\_UC_3$
= (10% * 0.6 + 10% + 0 * 0.4) * (30)
= 4.8 PHs

(4) E_{reuse}
= Effort of T_UC_1 + Effort of T_UC_2 + Effort of T_UC_3
= 8PHs + 21.6 PHs + 4.8 PHs = 34.4 PHs

(5) $E = E_{new} + E_{reuse}$
= 0 + 34.4PHs
= 34.4PHs

** Development Effort Calculation for Reuse Approach 2*

As in reuse approach 2, C_3 and C_4 will be reused by S, the software developers compare the use cases of S and those of C_3 and C_4. They get the values for use case gap metrics, as listed in table 5.

Table 5: Use Case Gap for Reuse Approach 2

	P_A	P_N	P_T	P_D
(T_UC_1, {R_UC_{31}, R_UC_{32}})	50%	20%	40%	10%
(T_UC_2, {R_UC_{41}})	80%	10%	20%	0
(T_UC_3, {})	/	/	/	/

Suppose the transformation factors $TF_1 = TF_2 = 0.6$ and $DF_1 = DF_2 = 0.4$, then the calculation steps for reuse approach 2's development effort is:

(1) Effort of T_UC_1
= $(P_{T1} * TF_1 + P_{N1} + P_{D1} * DF_1) * \sum$ Effort of $WP_j\_UC_1$
= (40% * 0.6 + 20% + 10% * 0.4) * (30+20)
= 24 PHs

(2) Effort of T_UC_2
= $(P_{T2} * TF_2 + P_{N2} + P_{D2} * DF_2) * \sum$ Effort of $WP_j\_UC_2$
= (20% * 0.6 + 10% + 0 * 0.4) * (25)
= 5.5 PHs

(3) E_{reuse}
= Effort of T_UC_1 + Effort of T_UC_2
= 24PHs + 5.5 PHs = 29.5 PHs

(4) E_{new} = Effort of T_UC_3 = 15 PHs, as estimated manually by software developers

(5) $E = E_{new} + E_{reuse}$
= 15 PHs + 29.5 PHs
= 44.5PHs

** Compare between Reuse Approach 1 and 2*

With the development effort calculation results gotten in previous steps, software developers can learn that reusing C_1 and C_2 for S will save 44.5 – 34.4 = 10.1 PHs effort than reusing C_3 and C_4. Although it's not the determinant factor for developers to select reuse approach 1, software developers can combine this factor with others (like the price for getting reusable components for each approaches, how developers skill availability for different approaches, and so on), to support more precise decision making.

3. Proof of Concept Implementation

3.1. High Level Architecture

As depicted by Figure 2, major components of the AREA support tool fall into four layers. Components in the representation layer provide software developers with the integrated interface for login, reuse approach design, candidate reuse approach tradeoff, and so on. This layer is implemented with JSP (JavaServer Pages)[16] and Struts[17], both widely available technologies.

Components in the application layer provide core functions, including user management, project management, alternative reuse approach management and estimation management. We use SCA (Service Component Architecture)[18] to implement the components in application layer.

Components in the data access layer provide convenient and stable database access interfaces for application layer components. iBatis[19] is used for the data access components implementation.

The repository implemented with DB2 provides persistent storage for user information, work product metrics, software project profiles, reuse approaches, etc.

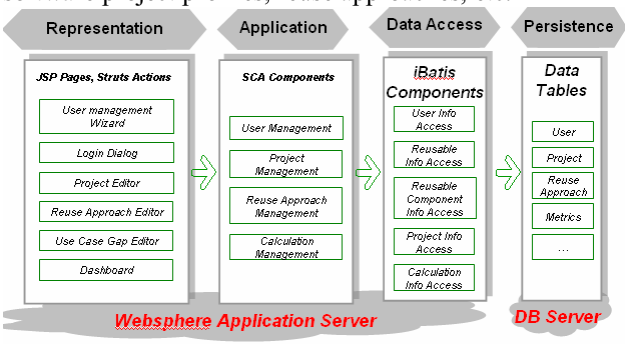


Figure 2: AREA Tool Architecture

3.2. Workflow

The architecture is designed to allow these functional components and tools to be accessed in the context of a collaborative workflow. Typically, a workflow would be started whenever new software is proposed for development. A project would be started to analyze the development effort for candidate reuse approaches. First a leading software developer would start a project for the to-be analyzed software, and assign other developers to work on the analysis using the Role assignment wizard. From that point, the leading software developer can monitor the project progress. The software developers assigned to different roles would perform the profiling and analytic tasks as prescribed in their role assignments. Figure 3 shows the screenshot for use case gap specifying task.

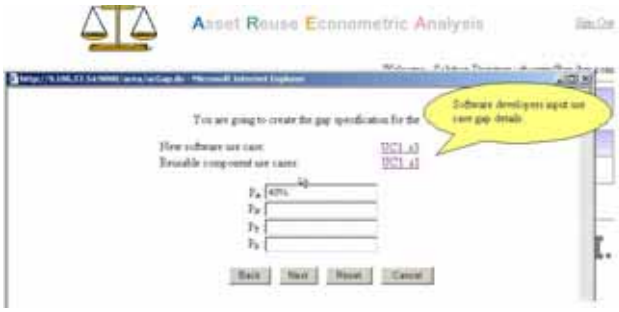


Figure 3: Specifying Use Case Gap

As prerequisite tasks are performed and completed, individuals assigned to dependent tasks would be notified that their tasks can and should be commenced. When the analytical tasks are completed, all the software developers can review the development effort calculation report and start reuse approach evaluation.

3.3. Dashboard

The tool provides an estimation effort dashboard to the software developers. The dashboard presents the components in each reuse approaches, the use case similarity relationship between new software use cases and reusable component use cases, the detail information about use case gap, the process and result of development effort calculation. Then, software developers can drill down into the details behind the effort result and understand the effort in a scientific way.

4. Pilot and feedback

Version 1.0 of the AREA tool is currently being piloted with solution development team working at IBM's Global Business Solutions Center (GBSC). The GBSC developers who use AREA tool comment that: in their previous reuse-based development practice, development effort estimation is not typically considered when selecting reusable components, as manual estimation of the development effort relies heavily on personal experience and different developers may give very different estimation. With the AREA tool and its backend method, the development effort calculation for candidate reuse approaches becomes more scientific and it's easier for them to come to an agreement on the result as the calculation process is explicit to all.

Also, the feedback from the pilot users drives new requirements for further method enhancement and tool development, which includes:

1) To extend the AREA method for considering Non-functional requirement gap between new software and reusable components while calculating the development effort.

2) To integrate the AREA tool with Rational Software Architect so that use cases, implementation work products,

and the traceability between them can be viewed in the native environment.

3) To allow customizable calculation for use gap P_A, P_N, P_T, and P_D based on basic use case actor, flow and extension point gap metrics.

4) To provide more intuitive and interactive reporting.

5. Conclusion and Future Works

In this paper, we present the AREA method, which enables systematic evaluation of development effort based on alternative reusable components selection. The core components of the method are a process guiding the evaluation and an algorithm for calculating the effort based on use case gap analysis between to-be developed new software and selected reusable components. A proof-of-concept implementation of AREA is built and being piloted with a solution development team working at IBM's Global Business Solutions Center (GBSC). Preliminary feedbacks reveal that the AREA tool and its backend method make the development effort calculation for candidate reuse approaches more scientific and it's easier for them to come to an agreement on the result.

In the future, we are going to enhance the method and tool according to suggestions acquired in present pilot, and continue the validation in more cases.

6. Acknowledgement

We thank Gopikrishnan Gopalakrishnan for leading the AREA trial and giving the feedback. We also thank Paul Borrel, William Tulskie and Liu Ying for illuminating discussions.

7. References

[1] I. Jacobson, M. Griss, P. Jonsson, Software Reuse: Architecture Process and Organization for Business Success, ACM Press, 1997.

[2] J.E. Gaffney, T.A. Durek, Software reuse—key to enhanced productivity: some quantitative models, Information and Software Technology archive, Volume 31, Issue 5, pp. 258-267, 1989.

[3] Crnkovic, I., Component-based Software Engineering–New Challenges in Software Development, 25th International Conference on Information Technology Interfaces Proceedings, IEEE Computer society, 2003, pp. 9-18.

[4] J.S Poulin, J. M. Caruso, and D. R. Hancock, The business case for software reuse, IBM Systems Journal, vol. 32, 4. pp. 567-594, 1993.

[5] G. Kotonya and A. Rashid, A strategy for Managing Risks in Component-based Software Development, 27th Eruomicro Conference 2001 Proceedings, IEEE Computer society, 2001, pp. 12-21.

[6] J. Kontio, OTSO: A Systematic Process for Reuseable Software Component Selection, CS-TR-3478, University of Maryland Technical Report, 1995.

[7] J. Kontio, A Case Study in Applying a Systematic Method for COTS Selection, 18th International Conference on Software Engineering Proceedings, pp.201-209, 1996.

[8] COTSRE: A COmponenTs Selection Method Based on Requirements Engineering, 7th International Conference on Composition-Based Software Systems, pp. 220-223, 2008.

[9] A. Carina, C. Jaelson, CRE: A systematic method for COTS components selection, XV Brazilian Symposium on Software Engineering (SBES), 2001

[10] http://ram.tap.ibm.com/com.ibm.ram/home.faces

[11] http://uddi.xml.org/uddi-org

[12] I.Jacobson,M. Christerson,et al. ,Object- Oriented Software Engineering: A Use Case Driven Approach, Addison - Wesley, 1992.

[13] G. Karner, Resource Estimation for Objectory Projects, http://www.bfpug.com.br/Artigos/UCP/Karner%20-%20Resource%20Estimation%20for%20Objectory%20Projects.doc, 1993.

[14] B. Anda, D. Dreiem, D.I..K. Sjoberg, and M. Jorgensen, Estimating Software Development Effort Based on Use Cases – Experiences from Industry, 4th International Conference on UML, pp. 487-502, 2001

[15] B. Anda, Comparing Effort Estimates Based on Use Cases with Expert Estimates, Proc. Empirical Assessment in Software Engineering, 2002.

[16] http://java.sun.com/products/jsp/

[17] http://struts.apache.org/

[18] http://www.ibm.com/developerworks/cn/webservices/ws-sca/

[19] http://ibatis.apache.org/

A Project Scheduling Method Based on Human Resource Availability

Lizi Xie [1,2], Junchao Xiao [1], Dapeng Liu [1,2], Qing Wang [1]
1 Institute of Software, Chinese Academy of Sciences
2 Graduate University of Chinese Academy of Sciences
{xielizi, xiaojunchao, liudapeng, wq}@itechs.iscas.ac.cn

Abstract

Organizational resource constraints should be taken into consideration when making a software project plan, as resource problems are one of the main causes of software project failures. Most of the resource schedule methods focus on optimizing resource allocation for a fixed project schedule. There is a lack of methods that guides project schedule based on resource constraints. Human resource is the most important resource in software development. We suggest a project schedule method that can satisfy both the schedule constraints among tasks and organizational human resource availability. It can provide decision support for project managers in making practical project schedule, such as helping them to schedule project more effectively to ensure high resource utilization rate, and providing useful information for project plan modification and resource plan creation.

1. Introduction

A practical project plan is an important assurance for project success. According to IEEE standards [1-2] for SPMP (Software Project Management Plan), project schedule and resource allocation are the most important parts of SPMP. The project would finish on time only has enough resource been assigned. Resource problem is one of the primary causes of software project failures [3]. In order to enhance organizational competitive edge and to ensure high ROI (Return on Investment), how to use resource more effectively is the key problem to solve for those large, global-distributed software companies.

In *CMMI for Development Version 1.2*, project resource includes "*labor, machinery/equipment, materials and methods*" [4]. Different from traditional industry processes, software development processes mainly depend on human capability. Human is the most important and complicated resource. William R. Tracey, in *Human Resources Glossary* defines Human Resources as: '*The people that staff and operate an organization*' [5]. The importance of human factors in software development gained more and more attentions [6]. How to take full advantage of human resource is the pivotal question for software companies.

In software project management field, how to set up, monitor and control project schedule effectively based on resource constraints remains a key problem. That human resource belongs to more than one departments in matrix organizations (IBM, HP, …), dynamic project teams and multi-project environment make it difficult to get a high utilization rate of human resource:

1 The low visibility of human resource availability makes project schedule according to resource status difficult.

2 The size of the project and the complexity of human resource make estimating the duration of the project consistent with human resource availability difficult.

We suggest a project schedule method based on organizational resource availability. The method first describes organization's human resource availability and defines human resource constraints for a task, then schedules tasks while assigning human resource to the tasks. Some evaluation indicators and analysis approach for the result are also offered to help project manager to solve resource problem and improve the project plan. Because organization's resource availability is considered during making project schedule, the final project schedule is practical which can guide the software development effectively by avoiding potential resource problem.

2. Related works

PERT (*Program Evaluation and Review Technique*) and CPM (*Critical Path Method*) [7] help project

manager to schedule tasks based on the constraints among tasks and estimate the time limit for the project. These methods are widely used in industrial fields, but they do not consider resource constraints when scheduling tasks, the resource utilization ratio can not be ensured. The derived critical path is often unpractical.

Critical-Chain [8-9] takes resource into consideration based on CPM. But it focuses on project schedule control, not the scheduling method.

Human resource is the core resource of software companies. How to describe human capability and manage human resource effectively is the key problem of software project management [3] [10]. It seems that most of the researches [11] on resource schedule methods do not consider the characteristic of human resource for optimization.

We proposed software process modeling based on organization entity capability (OEC-SPM) [12-15]. Organization entities with similar capability are modeled as a Process-Agent. Some descriptions of human resource in OEC-SPM are adopted in this paper.

The work in this paper stands on organizational resource availability. We believe that a project schedule which conforms to organizational resource availability is a strong assurance for software project success.

The rest of the paper is organized as follows: Section 3 describes the two kinds of constraint in project scheduling. The method will be explained in section 4. In section 5, an example is offered to illustrate the usage of our method. Section 6 discusses the conclusion and future work.

3. Schedule constraints and resource constraints

3.1. Basic concepts

Some concepts in our research scope are described as following:

(1) Role and role set

RS (Role Set) = { $Role_1$, $Role_2$, $Role_3$ ⋯}

Role is the description of capability and responsibility of human resource when executing a task. For example:

RS = {Designer, Programmer, Tester}

(2) Task and project

Task is the low level work package in WBS [1-2]. Its cost, workload, duration, work product and resource requirement can be easily recognized. We define task as follows:

T (Task) = {**TN**, **R**, **ED**, A**PR**, **ISD**, **IFD**, **PSD**, **PFD**, **HRA, NHRA**}

TN is the *task name*. R indicates that only the person who has the *role* can execute the task. ED is the *estimated duration* of the task. APR is the amount of estimated human resource required by the task. ISD is the *ideal start date* of the task without considering resource constraints. IFD is the *ideal finish date* of the task without considering resource constraints. PSD and PFD stand for the *plan start date* and *plan finish date* of the task after resource allocation. HRA is the *human resource list assigned* to the task. NHRA is the *number of human resource assigned* to the task.

For example, T = {module C development, Programmer, 7, 4, 2008-3-26, 2008-4-1, 2008-4-1, 2008-4-9, {"jack", "tom"}, 2} means the task called 'module C development' needs four programmers to work together for 7days. The ideal start date is 2008-3-26, The ideal finish date is 2008-4-1. The plan start date is 2008-3-28, and the plan finish date is 2008-4-3. The task has two human resources assigned, they are jack and tom.

A software project consists of many tasks.

SP (software project) = { $Task_1$, $Task_2$, $Task_3$ ⋯}

(3) Human resource

Task will be executed by the assigned human resource. It will run normally only when the assigned human resource has the capability needed and has enough time. We describe human resource as follows:

HRS (Human Resource Set) = { HR_1 , HR_2 , HR_3 ⋯}

HR (Human Resource) = {**N**, **RS**, **WC**}

N (Name) is the name of human resource. RS (Role Set) contains roles that human resource can act as. WC (Work Calendar) is the time table of human resource. It indicates which days are free and which are occupied by tasks.

WC (Work Calendar) = { Day_1 , Day_2 , Day_3 ⋯}

Day = {Date, Task}

Day.Task = null indicates that the human resource is free on that day. Day.Task ≠ null indicates that the work day is occupied by the task.

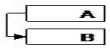
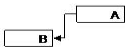
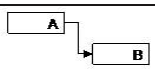
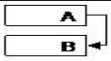
Assumption1: It is assumed that ED and APR of a task is reasonable. Task schedule and resource allocation will follow these estimated values. The analysis and evaluation of the result are also based on them.

3.2. Schedule constraints among tasks

Definition1. T.Pre-Tasks: Other tasks which can impact T' schedule are defined as **T.Pre-Tasks.**

Constraints among tasks mainly lie in schedule constraints because of work products. They can be easily recognized by project manager. T.Pre-Tasks can be empty, which means the task is independent. When T.Pre-Tasks are not empty, each task in T.Pre-Tasks may have four kinds of schedule constraints (Table 1) with T. They are SS (start-start), SF (start-finish), FS (finish-start) and FF (finish-finish).

Table 1. Four kinds of schedule constraints

Type	Explanation	Figure
SS	B.PSD>=A.PSD	
SF	B.PFD>=A.PSD.	
FS	B.PSD>=A.PFD	
FF	B.PFD>=A.PFD.	

T.Pre-Tasks can be divided into four sub sets based on the four constraint types. They are: **T.Pre-Tasks.SS**, **T.Pre-Tasks.SF**, **T.Pre-Tasks.FS**, and **T.Pre-Tasks.FF**. The tasks in each sub set have the same type of schedule constraint with T. The schedule of T should satisfy the following four SC (schedule constraints):

SC1. SS-All: T.PSD >= T.Pre-Tasks.SS.MPSD
MPSD (Max Plan Start Date) is the latest plan start date of all the tasks in the given task set.
SC2. SF-All: T.PFD >= T.Pre-Tasks. SF.MPSD
SC3. FS-All: T.PSD >= T.Pre-Tasks.FS.MPFD
MPFD (Max Plan Finish Date) is the latest plan finish date of all the tasks in the given task set.
SC4. FF-All: TPFD >= T.Pre-Tasks.FF.MPFD

Task schedule result must satisfy the four constraints above (SC1-SC4), otherwise the project execution will be affected by schedule conflicts.

Definition2. T.ISD: The later date of T.Pre-Tasks.SS.MPSD and T.Pre-Tasks.FS.MPFD is called **T.ISD** (ideal start date).

Definition3. T.IFD: The later date of T.Pre-Tasks. SF.MPSD and T.Pre-Tasks.FF.MPFD is called **T. IFD** (ideal finish date).

Definition4. T.SD: The number of days between T.ISD and T.PSD is called **T.SD** (schedule delay). When T.SD > 0 we say that T is delayed.

3.3. Resource constraints of task

Project schedule should satisfy resource constraints besides the four schedule constraints. When assigning human resource to tasks, three RC (Resource constraint) should be considered.

RC1: T.R ∈ HR.RS
RC2: ∀ HR ∈ { HR | HR.WC.Day.Date ∈ [T.PSD, T.PFD]} → HR.WC. Day.Task = null
RC3: T.APR =T.NHRA

RC1 means that the assigned human resource must have the ability of the desired role. RC2 means that the assigned human resource must have enough spare time in the given duration of the task. Humans who satisfy RC1 and RC2 can be assigned to the task. When the human resources of the task satisfy all the three constraints, the schedule and resource of the task can be decided.

Definition5. HR.NET (Number of Executable Tasks): The number of un-assigned tasks which satisfy RC1 and RC2 with HR is called HR.NET.

4. Project schedule method based on resource availability

4.1. Method of resource allocation and project schedule

The main framework of the method is shown in figure 1.

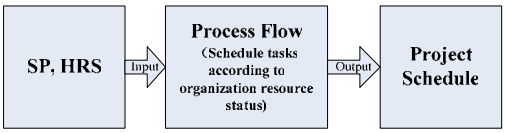


Figure 1. Main framework of the method

The detailed process flow is shown in figure 2. When scheduling tasks, we try to let the tasks start as early as possible, and we have an assumption below considering the characteristic of human resource.

Assumption2: We assume that it is a strong assurance for high productivity that the human does the same job continuously. Too much alternation among different tasks will lead to low work efficiency.

Inputs: Task set (SP), Human resource set (HRS)
Process flow:
Step1: Get task set (ts) = {T | T ∈SP ∧ T.Pre-Tasks = Φ};
Step2: Call Allocation (ts);
Step3: Get task set (ts) = {T | T ∈SP ∧ T.Pre-Tasks ≠Φ∧ {M|M∈T.Pre-Tasks ∧ M.HRA = null} = Φ∧ T.NHRA=null};
Step4: Call Allocation (ts);
Step5: if {T | T ∈ SP∧T.HRA = null}≠Φ, go to step 3; else go to step 6;
Step 6: Allocation finish.
The detailed process flow of some core functions are described in pseudo codes as following:

Function Allocation (ts)

```
{
        get all tq of ts (tq is a permutation of ts );
        for each tq
        {
                copy HRS to tq.HRS;
                for each task T  in tq
                {
                        Call Process(T, tq.HRS);
                }
                Re-calculate ISD and IFD of the tasks whose HRA is null in SP;
                get the plan finish date of project which is recorded as PFDP;
        }
        Choose the tq which has the earliest PFDP;
        Update SP and HRS according to the chosen tq and tq.HRS;
        }
}
```

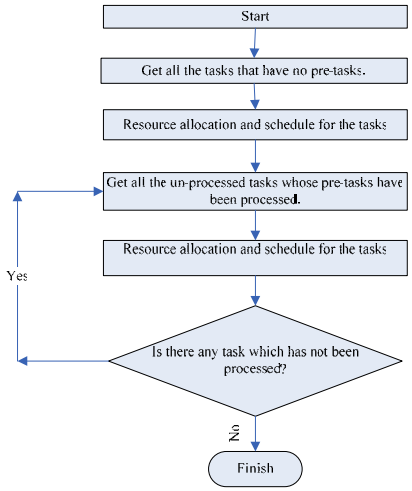


Figure 2. The detailed process flow

Function Process (T, HRS)

```
{
Get the number N of humans who have the role which T requires;
Set the smaller one of N and T.APR as the target resource amount Y;
Update HR.NET for all humans in HRS;
Ordered all the HR in HRS by ascending HR.NET;
T.PSD = T.ISD;
Do
{
        T.PFD = getPFD(T);
        Search the human combination which can satisfy RC1, RC2;
        If the size of human combination = Y
        {
                Break;
        }
        T.PSD = T.PSD +1;
} while (true);
T.SD = T.PSD – T.ISD;
Update T.HRA;
Update the work calendar of the human in HRS;
}
```

Function getPFD (T)

```
{
If (T.PSD+T.ED>=T,IFD) return T.PSD+T.ED;
If (T.PSD+T.ED<T.IFD) return T.IFD;
}
```

After all the tasks have been processed, we can get a practical project schedule which accords with schedule constraints and organizational resource availability.

4.2. Indicators for result evaluation

In order to evaluate and analyze the results, we define three indicators as following:

Indicator 1: **RUT** (Resource Utilization rate of the Task) = ED / (PFD – PSD + 1).

RUT>1 indicates that human resources of the task are overloaded. RUT<1 indicates that the human resource may have much more spare time during executing the task.

Indicator 2: **RST** (Resource Satisfaction rate of the Task) = NHRA / APR.

RST<1 indicates that the resource is not enough for the task.

Indicator 3: **DTP** (Delayed Task rate of the Project) = Number of tasks whose SD > 0/ Number of tasks in the project.

DTP > 0 means that there are tasks which can not start in time because of resource limitation.

These indicators can help project manager evaluate the result and find the root cause of abnormity. For example, if RUT<1, the project manager could assign additional work to the human resources to raise the resource utilization rate. If RUT>1, the project manager has to modify ED and APR of the task to avoid schedule risk. If RST<1 or DTP>0, new employees might be employed, or let the relative human resource delay less important tasks.

5. Example

5.1. Case design

Take a typical small software project for example. The plan start date of the project is assumed as 2008-4-1. The project manager wants to know the probable

finish date of the project when considering resource status. The software is divided into two modules (A and B). The role set is defined as RS = {D (Designer), P (Programmer), T (Tester)}. The tasks are listed in Table 2. The schedule constraints among tasks are shown in Table 3. The human resource status is shown in Table 4. Holidays are not considered for simplification.

Table 2. Tasks in the project

ID	Name	R	ED	APR
T1	System design	D	14	3
T2	A detailed design	D	10	2
T3	A development	P	6	2
T4	A test	T	4	2
T5	B detailed design	D	10	2
T6	B development	P	7	2
T7	B test	T	4	2
T8	Integration test	T	10	3

Table 3. Schedule constraints among tasks

ID	Pre-Tasks and SC	ID	Pre-Tasks and SC
T1	Null	T5	T1-FS
T2	T1-FS	T6	T5-FS
T3	T2-FS	T7	T6-SS, T6-FF
T4	T3-SS, T3-FF	T8	T4-FS, T7-FS

Table 4. Human resource availability

ID	RS	WC (Days occupied by other projects)
P1	D P	2008-05-05---2008-05-14
P2	D P	2008-04-10---2008-04-20
P3	D P T	2008-05-04---2008-05-10
P4	D P T	2008-04-25---2008-04-26
P5	T	2008-04-15---2008-04-23
P6	T	2008-05-15---2008-05-25

5.2. Resource allocation and task scheduling

At first, we schedule tasks considering only the schedule constraints. After implementing the method in section 4.2, we get the scheduling result show in Figure 3.

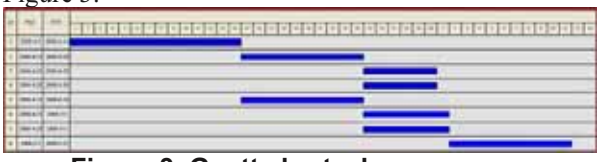


Figure 3. Gantt chart when resource constraints are not considered

The project duration is 41 days which is perfect because all tasks can start as early as possible. The project will finish on 2008-5-11, but it is impractical because resource constraints were not considered.

Now we implement our method to this case. First we will process tasks which have no Pre-Tasks. T1.ISD=2008-4-1, T1.IFD=2008-4-14. The satisfying resource combination is {P1,P3,P4}. T1.PSD=2008-4-1, T1.IFD=2008-4-14. Then we will process other tasks which have no unscheduled Pre-Task. {T2, T5} is our next target. We find that {T2, T5} can't start synchronously because there is not enough resource available. We will decide to delay which task using the method in section v4.1. The task set has two permutations: tq1={T2, T5} and tq2={T5, T2}. The schedule results of the two different permutations are shown in Table 5.

Table 5. Two optional schemes

	T	PSD	PFD	Human
{T2,T5}	T2	2008-4-15	2008-4-24	P1 P3
	T5	2008-4-25	2008-5-5	P1 P2
{T5, T2}	T2	2008-4-25	2008-5-5	P1 P2
	T5	2008-4-15	2008-4-24	P1 P3

We will calculate the ISD and IFD of the other tasks based on the two results. The finish date of the project is 2008-5-21 for tq1, and 2008-5-20 for tq2. We will choose tq2 to continue our approach.

{T3, T6} is the next target. Schedule the tasks in the same way until all the tasks have been processed. Finally the project schedule result is shown in Figure 4, and the task attributes are shown in Table 6.

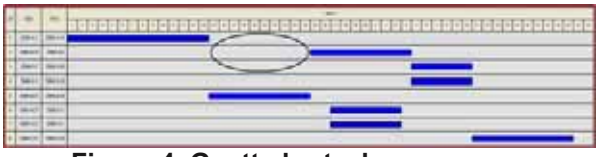


Figure 4. Gantt chart when resource constraints are considered.

Table 6. Task attributes after resource allocation

ID	PSD	PFD	HR	SD
T1	2008-04-01	2008-04-14	P1 P3 P4	0
T2	2008-04-25	2008-05-04	P1 P2	10
T3	2008-05-05	2008-05-10	P2 P4	0
T4	2008-05-05	2008-05-10	P5 P6	0
T5	2008-04-15	2008-04-24	P1 P3	0
T6	2008-04-27	2008-05-03	P3 P4	2
T7	2008-04-27	2008-05-03	P5 P6	0
T8	2008-05-11	2008-05-20	P3 P4 P5	0

Different from the schedule in Figure 3, the final duration of the project has been extended to 50 days. The overall schedule is longer but more practical because it can guarantee that all the tasks can be executed with enough resource. The black cycle in Figure 4 indicates the root cause of schedule delay. T2 will be delayed for 10 days because of limited human resource. Also, both T6 and T7 can not start as early as we expected in Figure 3.

5.3. Result analysis

We can use the three indictors in section 4.2 to analysis the schedule result. Indictors for tasks are shown in Table 7. The DTP is 2/8=0.25. From the value of the indictors two problems are discovered.

Table 7. RUT and RST of each task

ID	RUT	RST	ID	RUT	RST
T1	1	1	T5	1	1
T2	1	1	T6	1	1
T3	1	1	T7	0.57	1
T4	0.67	1	T8	1	1

(1) RUT < 1: The RUT of T4 and T7 is below 1. Project manager could assign additional tasks to P5 and P6 during the execution of task T4 and T7.

(2) DTP > 0: The causes of this abnormity are task T2 and T6 (T2.SD=10, T6.SD=2). That P2 does not have enough time during 2008-4-15 and 2008-4-20 causes T2 could not start in time. We will not discuss T6 because it is not on the critical path now.

Project manager may wonder whether hiring a new designer is necessary. Supposing a new designer P7 is added to the HRS, after implementing our method, we could get the schedule result shown in Figure 5, and the task attributes are shown in Table 8.

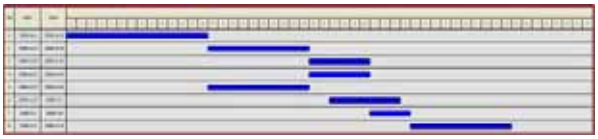


Figure 5. Gantt chart with a new designer

Table 8. Task attributes with a new designer

ID	PSD	PFD	HR	SD
T1	2008-04-01	2008-04-14	P1 P3 P7	0
T2	2008-04-15	2008-04-24	P1 P3	0
T3	2008-04-25	2008-04-30	P1 P2	0
T4	2008-04-25	2008-04-30	P5 P6	0
T5	2008-04-15	2008-04-24	P4 P7	0
T6	2008-04-27	2008-05-03	P3 P4	2
T7	2008-05-01	2008-05-04	P5 P6	4
T8	2008-05-05	2008-05-14	P4 P5 P6	0

The project duration would be 44 days after hiring a new designer. The different result will help the project manger to make decisions. After considering all the other factors the project manager can decide whether hiring a new man or delaying some assigned tasks of the key human resources, or that even 50 days is acceptable.

The case is simple but the usage of our method is illustrates clearly. It is proved that our method can provide decision support for project managers in many aspects, such as making practical project schedule, reaching high resource utilization rate, optimizing project plan and generating human resource plan.

6. Conclusion and future work

In this paper, we propose a method for project scheduling and resource allocation with consideration of both schedule constraints and resource constraints. Our method can offer decision support for project managers when making practical project plan. It can help software companies to achieve higher resource utilization rate.

Future work will focus on assistant tools development. The method will be integrated with Process-Agent technique. Optimized human resource allocation method using simulation technique will be further researched.

Acknowledgments: This work is partially supported by the National Natural Science Foundation of China under grant Nos. 60573082, 60473060, 90718042, the Hi-Tech Research and Development Program (863 Program) of China under grant No. 2006AA01Z185, 2007AA010303, as well as the National Basic Research Program (973 Program) of China under grant No. 2007CB310802.

7. References

[1] IEEE Std 1058-1998, IEEE Standard for Software Project Management Plans

[2] IEEE/EIA 12207.1-1997, IEEE/EIA Guide for Information Technology-Software life cycle processes-Implementation considerations.

[3] Kathy schwalbe, “Information technology project management”, 2e, Thomson Learning

[4] CMMI for Development, Version 1.2. Software Engineering Institute, 2006

[5] William R. Tracey: “The Human Resources Glossary”, Saint Lucie Pr

[6] Tom DeMarco and Timothy Lister,: “Peopleware: Productive Projects and Teams”, 2e. Dorset House Publishing Company, 1999.

[7] Harold kerzner: “Project management: a system approach to planning, scheduling, and controlling”. 9e

[8] Goldratt E.M,: “Critical Chain”. The North River Press Publishing Corporation .Great Barrington, 1997.

[9] Leach L P : “Critical chain Project Management” , Artech House Professional Development library.2000

[10] PMI, Project Management Body of Knowledge 2004

[11] Brucker P: “Scheduling Algorithms”. Springer Verlag. 2001.

[12] X. Zhao: “An Agent-Based Self-Adaptive Software Process Model”. Journal of Software, 2004.03, Vol. 15, No. 3, Mar. 2004

[13] Q. Wang: “Software Process Management: Practices in China”. M. Li, B. Boehm, and L.J. Osterweil (Eds.): SPW 2005, LNCS 3840, pp. 317-331

[14] Qing Wang: “A Process-Agent Construction Method for Software Process Modeling in SoftPM”. Q. Wang et al. (Eds.): SPW/ProSim 2006, LNCS 3966

[15] Lei Zhang: “A Tool to Create Process-Agents for OEC-SPM from Historical Project Data”, ICSP2007, LNCS 4470, pp.84-95

Estimating the Effort of Independent Verification and Validation in the Context of Mission-Critical Software Systems – A Case Study

Haruka Nakao[a], Adam Trendowicz[b], Jürgen Münch[b]
[a] *Japan Manned Space Systems Corporation, Ibaraki, Japan*
[b] *Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany*
haruka@jamss.co.jp, {Adam.Trendowicz, Juergen.Muench}@iese.fraunhofer.de

Abstract

The ability to generate sufficiently accurate effort estimates can be seen as a key success factor for multi-organizational projects focusing on the development of large and critical software systems. This is caused, for instance, by the need for synchronizing multiple development and verification and validation processes. Paradoxically, effort predictions in the critical software systems domain are still relying on human judgment. This requires much overhead and its reliability depends largely on the expertise and individual preferences of the involved experts. In particular, verification and validation by independent entity (IV&V) needs an estimation method that supports negotiating and managing IV&V costs in the context of sparse measurement data and low availability of domain experts. In order to address these problems, in this paper we propose applying a hybrid effort estimation method called CoBRA® for estimating effort for IV&V of mission-critical software systems. When applied in an industrial context, CoBRA® improved estimation accuracy and precision by about 40%, on average, compared to experts estimates and OLS regression.

1. Introduction

The average company spends about 4 to 5 percent of its revenue on information technology, with those that are highly IT-dependent - such as financial and telecommunications companies - spending more than 10 percent on it [5]. Now, a great part of those investments is wasted because software organizations are still proposing unrealistic software costs, work within tight schedules, and, in consequence, finish their projects behind schedule and budget (about 50% of projects), or do not complete them at all (more than 25% of projects). Moreover, even though projects are completed within a target plan, the functionality and quality of products delivered are usually cut to fit this plan [11]. This indicates that software project planning is a critical success factor of a software project.

Project planning in the safety-critical domain is particularly important and difficult at the same time. Large functional constraint, high quality requirements, and involvement of several independent parties make it much more challenging to plan mission-critical projects than to plan ordinary, non-critical software development projects. In that context, effective synchronizing activities of all involved parties are a key factor for project success. One example activity requiring such synchronization is verification and validation done by an independent organization, or independent verification and validation (IV&V). A mismatch between the IV&V plan and the overall project plan may lead to significant delays or (in extreme cases) to the skipping of certain IV&V activities. In consequence, the whole mission might be exposed to the high risk of a significant loss of money, and, in the worst case, injury or death of people. Yet, comprehensive support for planning and managing IV&V is missing.

Effort estimation approaches proposed by the research community have traditionally focused on planning and tracking classical, in-house software development. Effort estimation methods that grew upon those objectives focus on providing exact estimates. They do not, however, support an easily understandable, systematic and reliable analysis of the most relevant causal effort dependencies. Even though an accurate prediction is provided, software practitioners have hardly any support to prevent potential project overruns. In the short-term perspective, this would mean a lack of a solid basis for effectively mitigating project risks, and in the long-term perspective, a limited ability to identify process improvement areas and to learn. Moreover, estimation methods promoted by the research community require large data sets, whereas methods commonly employed by industry extensively involve domain experts.

All those aspects significantly reduce the applicability of existing estimation methods in the IV&V context, where reliable and comprehensive project management has to be provided despite the minimal availability of quantitative data and human expertise.

In this paper, we propose applying the Cost Estimation, Benchmarking, and Risk Analysis method (CoBRA®) [12][17] to estimate the effort of the IV&V of mission-critical software systems. CoBRA® is a hybrid method that combines analytical and expert-based estimation. It provides a systematic way to transform various sources of organizational knowledge (minimal set of measurement data and expert judgment) into a transparent and reusable effort model that supports achievement of a variety of project management objectives, such as risk management or nego-

tiating of project costs.

The remainder of the paper is organized as follows: Section 2 briefly characterizes the IV&V context. Section 3 gives an overview of existing software effort estimation methods, followed by a more detailed description of the CoBRA® method in Section 4. Section 5 presents the empirical results of an industrial application of CoBRA® for planning the effort of IV&V, followed by lessons learned (Section 6) in the study. The paper ends up with a brief summary and conclusions given in Section 7.

2. Independent Verification and Validation

2.1. Characteristics of IV&V

Independent verification and validation (IV&V) can be defined as a process where software work products generated by a development team are verified and validated by a completely independent organizational entity. Independence is considered here [7] in terms of technical, managerial, and financial independence. IV&V is typically applied in the context of safety- or mission-critical software systems, such as space and nuclear plant systems.

The typical constraint of IV&V, as compared to classical in-house V&V, is limited information on processed artifacts. On the one hand, there is limited knowledge about the software development environment; on the other hand, IV&V has to handle various types of mission-critical systems. This variety does not allow for collecting many historical project data. Moreover, involvement of three sites (customer-, development-, and IV&V-entity) in the software development process contributes to frequent and unpredictable requirements change.

In that context, managing an IV&V project's resources is critical and difficult at the same time.

2.2. Objectives of Effort Estimation

Besides traditional estimation objectives such as precise planning and tracking software resources, project needs decision-making support. In particular, explicit identification of factors having the greatest impact on IV&V cost and their quantitative impact should be supported. Identification of customer-specific factors (e.g., level of customer support) may be used to justify and negotiate IV&V costs that cannot be influenced by the IV&V supplier. On the other hand, identification of the IV&V supplier's characteristics (process and human capabilities) that have the greatest impact on increased IV&V costs will allow targeting improvement actions to specific process areas and improving the efficiency of IV&V. In consequence, project risks can be mitigated timely and critical organizational processes can be improved.

2.3. Current Effort Estimation Practices

A survey about current estimation practices at Japan Manned Space Systems Corporation (JAMSS) revealed that measurement data from around 10 already completed projects have been collected within the past 10 years. However, the data suffered from significant incompleteness (around 20% of missing data) and large variability – due to the high uniqueness of the considered projects. Since hardly any data-driven estimation method that would meet the estimation objectives (Section 2.2) can be applied reasonably, estimates are typically based on the judgment of one or more domain experts. Although sparse project data are available, experts based their estimates solely on personal experiences. One of the reasons is that the available simple size measures, such as pages of software requirements document, are believed not to reflect the amount of IV&V effort reliably. Yet, expert-based estimation did not provide satisfactory support for project management. First, the reliability of the estimates depends largely on individual expertise and preferences of involved domain expert. In consequence effort estimates are not accurate and vary widely across projects (see Table 4 and Table 5 in Section 5.5). Moreover, estimation costs much effort each time it is performed, and since it does not provide any explicit effort model, it hardly supports decision making in a project.

3. Related Work

Numerous types of estimation methods have been developed over the last decades. In this section we provide a brief overview of existing estimation methods from the viewpoint of their applicability in the context of IV&V. For a comprehensive review and comparative evaluation of existing methods, please refer to [16].

Existing effort estimation methods differ basically with respect to the type of inputs they require and the form of the estimation model they do provide. With respect to input data, we differentiate between three major groups: data-intensive, expert-based, and hybrid methods (combining available data and expert knowledge in order to come up with estimates). Among the data-intensive methods, some require past project data for building customized models (*define-your-own-model* approaches), others provide an already defined model, where factors and their relationships are fixed based on a set of multi-organizational project data (*fixed-model* approaches). The major advantage of fixed-model approaches is that they, theoretically, do not require any historical data to be applied. Those methods might be especially attractive in the IV&V context, where very sparse (if any) data are typically available. Yet, in practice, fixed models, such as COCOMO [2][1], are developed for a specific context (typically different from IV&V) and are, by definition, only suited for estimating the types of projects for which the fixed model was built. The applicability of such models for the IV&V context is, in practice, very limited. In order to improve their performance, a significant amount of organization-specific project data would be required for calibrating the generic model. In that case, the

little usefulness of the fixed-model approaches for IV&V effort estimation would not differ much from the define-your-own-model approaches, which require a significant amount of context-specific data to build customized effort models [15]. Application of the define-your-own-model methods in the context of IV&V is further limited by the additional requirements of specific methods. Parametric approaches, such as regression [14], for instance, make several assumptions about underlying project data (completeness, normal distribution, etc.) that are rarely met in the software domain. Non-parametric methods originating from the machine learning domain, such as artificial neural networks (ANN) [3] or Decision Trees/rules [15], make practically no assumptions about the data but are quite sensitive to their parameter configuration and there is usually little universal guidance regarding how to set those parameters. Thus, finding appropriate parameter values requires some preliminary experimentation.

In contrast to data-intensive methods, *expert-based estimation* does not require any project measurement data because estimates are based on the judgment of one or more human experts [2]. Expert estimation is, in fact, commonly used in the software industry (including IV&V). It does, however, have several significant limitations. First, much effort is required each time estimation is performed, and the reliability of the outputs it provides largely depend on the expertise and individual preferences of the human experts involved. Moreover, since the rationale underlying final estimates is not modeled explicitly, there is hardly any support for effective decision making in a project (risk management, process improvement, project scope negotiations, etc.). Recently, a few hybrid methods have been proposed to cope with deficits of data-intensive and expert-based estimation. They combine a reduced amount of both measurement data and human expertise to provide more reliable estimates with limited estimation overhead. Empirical applications [17][10] report on their higher estimation accuracy and stability when compared to data- or expert-based methods. Moreover, methods that employ explicit causal effort modeling (e.g., CoBRA® [17]) have proven to greatly contribute to the achievement of a variety of organizational objectives, such as risk management or process/productivity improvement.

4. The CoBRA® Method

CoBRA® [12][15] is a hybrid method combining data- and expert-based effort estimation approaches. CoBRA® the method is based on the idea that project effort consists of two basic components: nominal project effort and an effort overhead portion as presented below.

Nominal effort is the effort spent only on developing a software product of a certain size in the context of a hypothetical "ideal" project that runs under optimal conditions; i.e., all project characteristics are the best possible ones ("perfect") at the start of the project. *Effort overhead* is the additional effort spent on overcoming the imperfections of a real project environment, such as insufficient skills of the project team. In this case, a certain effort is required to compensate for such a situation, e.g., team training has to be conducted. In CoBRA®, effort overhead is modeled by a causal effort model that consists of factors affecting project effort within a certain context. The causal model is obtained through expert knowledge acquisition (e.g., involving experienced project managers).

$$\text{Effort} = \underbrace{\text{Nominal Productivity} \cdot \text{Size}}_{\text{Nominal Effort}} + \text{Effort Overhead} \quad (1)$$

$$\text{Effort Overhead} = \sum_i \text{Multiplier}_i(\text{Effort Factor}_i) + \sum_i \sum_j \textit{Multiplier}_{ij}(\text{Effort Factor}_i, \text{Indirect Effort Factor}_j) \quad (2)$$

An example is presented in Figure 1. The solid and dashed arrows indicate direct and indirect relationships, respectively. For instance, *Requirements volatility* has a direct impact on development effort. The strength of this negative influence on effort may, however, be modified (compensated) by *Disciplined requirement management* (indirect influence). The effort overhead portion resulting from indirect influences is represented by the second component of the sum shown in (2).

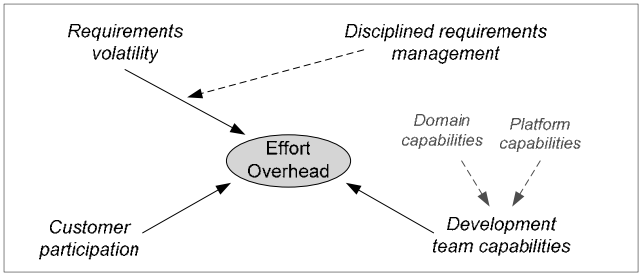


Figure 1: Example of a Causal Effort Model

The influence on effort and between different factors is quantified for each factor using experts' evaluation. The influence is measured by means of effort overhead, i.e., a relative percentage increase of the effort above the nominal project. In order to capture the uncertainty of evaluations, experts are asked to give three values: the maximal, minimal, and most likely cost overhead for each factor (triangular distribution).

The second component of CoBRA®, the nominal project effort, is based on data from past projects that are similar with respect to certain characteristics (e.g., development type, life cycle type) that are not part of the causal model. These characteristics define the context of the project. Past project data is used to determine the relationship between cost overhead and costs (see equation #1). Since it is a simple bivariate dependency, it does not require much measurement data. In principle, merely project size and effort are required, whereby both can be measures using any valid metric representing project size and effort.

Based on the quantified causal model, past project data, and current project characteristics, an effort overhead model is generated using a simulation algorithm (e.g., Monte Carlo). The probability distribution obtained could be used further to support various project management activities, such as effort estimation, evaluation of effort-related project risks, or benchmarking. More details regarding the CoBRA® method can be found in [12][15].

5. Case Study

5.1. Context of the Study

The study was performed in the context of JAMSS, a company that performs IV&V of space software systems (embedded software domain). JAMSS has been, for instance, supporting IV&V for critical space software systems created by the Japan Aerospace Exploration Agency (JAXA) for more than 10 years.

In this study, we focused on IV&V of the software requirements specification documents using the *document review* technique. The document review process starts with a risk analysis to identify a software system's operational risks. Software requirements are then reviewed in more detail based on their operational risks with respect to one or more review objectives. In principle, there were six review objectives [8] (Table 2): (O1) risk analysis, (O2) state transition completeness and consistency, (O3) design completeness for exceptional behavior, (O4) timing correctness and consistency, (O5) interface correctness and consistency, and (O6) traceability.

There were three ***domain experts*** involved in the study (Table 1) who provided their knowledge to build the effort overhead model. The main fields of expertise covered by involved experts included: software product quality & safety assurance (SPQSA), software safety reviews (SR), and safety assurance in operation (SAO).

Table 1. Involved domain experts

Expert	Expertise	Domain experience [#years]	Estimation experience [#projects]
1	SR	7	8
2	SPQSA	8	9
3	SAO	4	6

As project measurement data, the number of document pages was selected as the size of a software requirement because even if the complexity of a requirement complexity is related to the effort for reviewing a document, the document itself has to be read by the IV&V team in order to find out what this complexity is.

IV&V effort data from five projects were collected for each project. In practice, because some IV&V objectives were not addressed, effort data were not collected for each IV&V objective except for one project. Therefore, weekly working statuses of IV&V were used to abstract the effort for each IV&V objective. ***Measurement data*** available for the estimation included size and effort. Size was measured in pages of software requirements for objectives O1 to O5 and system specification (software and hardware) for objective O6 additionally. The effort was measured in person-days (PD).

Table 2. Review objectives considered in the study

Id	Objective	#projects
O1	Risk analysis	5
O2	State transition completeness/consistency	5
O3	Design completeness	5
O4	Interface completeness/consistency	4
O5	Timing consistency/correctness	3
O6	Traceability with correctness	5

5.2. Study Objectives

The objective of the study was to validate accuracy and precision of CoBRA in the context of JAMSS IV&V (compared to expert judgment and Ordinary Least Squares method) and its contribution to the achievement of defined organizational objectives (Section 2.2).

5.3. Study Design

5.3.1. Effort Estimation Procedure

Motivated by its numerous benefits, the CoBRA® Method was proposed as best fitting the effort estimation capabilities and objectives of JAMSS. First of all, CoBRA® proposes a systematic way to build an explicit and reusable effort model based on both implicit knowledge of domain experts and sparse measurement data. Moreover, it provides on the output a transparent and intuitive model of causal effort dependencies specific for the context where it was applied. The first step of the effort estimation procedure included development of the CoBRA® model using the knowledge of the involved domain experts and measurement data (size and effort) from already completed (historical) projects. For each of the six IV&V objectives specified in the study (Table 2), a separate CoBRA® model was developed. After the CoBRA® models had been created, each was validated on the historical data in a leave-one-out cross-validation experiment.

5.3.2. Study Hypotheses

In order to effectively support achievement of the estimation objectives, the outputs of CoBRA® need to be reliable. In our study, we evaluate reliability byvalidating the predictive performance of the estimation outputs, measured in terms of predictive accuracy and precision. We expect that CoBRA® will outperform the currently employed expert-based estimation as well as the Ordinary Least Squares method (OLS), one of a few data-driven methods that are applicable in the study context (due to very sparse measurement data). This leads us to two study hypotheses:

H1. *CoBRA® provides more accurate and more precise estimates than estimation based on expert judgment.*

H2. *CoBRA® provides more accurate and more precise estimates than estimation based on OLS.*

5.3.3. Evaluation of Estimation Performance

The effort models created in the study effort models were evaluated with respect to their predictive performance. We define *predictive performance* as the ability of the effort model to provide accurate and precise estimates. *Estimation accuracy* refers to the nearness of an estimate ($\hat{E}$) to the true value (E). In order to remain comparable to other estimation studies, we use common estimation error measures and accuracy measures [4], such as *relative error* (*RE* in equation #3) and *mean magnitude of relative error* (*MMRE*).

$$RE_i = \left(\hat{E}_i - E_i\right) / E_i \tag{3}$$

The Conte's RE and MRE measures are the subject of common criticism in the software research community [6]. One of the alternative measures of estimation error proposed is the so-called *z* measure (equation #4) [6]. It quantifies the ratio of the estimate to the actual value

$$z_i = E_i / \hat{E}_i \tag{4}$$

Estimation precision refers to the degree to which several estimates are very close to each other (i.e., the scatter in the data). For the purpose of comparability to other studies, we adopt the *Pred.m* measure. The *Pred.m* measures the percentage of estimates that are within *m%* of MRE [4]. In our study, we use *m = 25%* as typically employed in software estimation studies. Moreover, we adopt *relative standard deviation* (*RSD*) (5) proposed by [6] for software effort estimation as uncorrelated with size (S_i) (which is a weakness of classical standard deviation measures).

$$RSD = \sqrt{\frac{\sum_{i=1}^{n}\left[\left(E_i - \hat{E}_i\right)/ S_i\right]^2}{n-1}} \tag{5}$$

5.4. Study Execution

During the study execution, six CoBRA® models were created for each of the IV&V objectives.For each model, domain experts identified several factors (Table 3) that are responsible for the variance of IV&V efficiency for a certain objective (Figure 2).

5.5. Results and Interpretation

This section presents the results of the empirical study. Table 4 and Table 5 present the aggregated measures of the predictive performance. In order to test the significance of the observed effects, appropriate statistical test were performed [13] (at $\alpha = 0.05$). The results of a *Shapiro-Wilk W* test indicated that the MRE and z results come from normal population; in that case, a parametric *Paired T-test* for homogeneity of means was used. Since the RSD data violated the normality assumption, a non-parametric *Wilcoxon Matched-Pairs Signed Rank* test was used. Note that expert estimates were available for a subset of the past projects considered (indicated as *n* in Table 4 and Table 5). Finally, as we were afraid that for such a small data sample, statistical tests would not have enough power ($\beta\text{-}1 \geq 80\%$), we performed a power analysis.

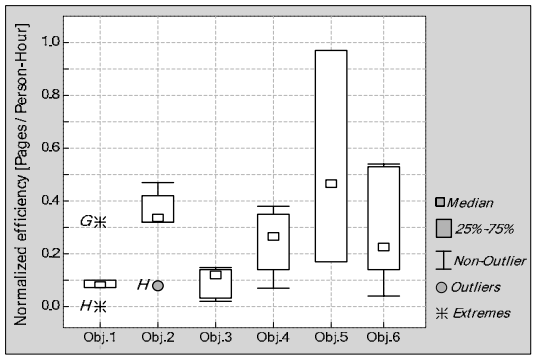


Figure 2. Overall efficiency of IV&V

Table 3. Effort factors considered in the study

Factors influencing IV&V efficiency	Objectives
Domain experience of the IV&V team	O1-O3
Requirements volatility allowed within an initial contract	O2-O4, O6
Novelty of applied IV&V technique	O3
Number of system's interfaces to other (sub)systems	O4, O6
Time pressure in the last IV&V phase	O5
Level of risk assessment done by a supplier or customer	O1
Fault Tree Analysis done by IV&V company	O1
Timing consistency objective included in IV&V	O5
Field Programmable Gate Array (FPGA) review performed	O5
New (inexperienced) personnel involved in IV&V	O1

5.5.1. Hypothesis H1

The results of the empirical investigation (Table 4) suggest that, in principle, CoBRA® provides noticeably more accurate estimates than either expert judgment or OLS for all considered IV&V review objectives (a few exceptions are marked in gray). For example, it improves MMRE by 70% and 40%, on average, compared to OLS and domain experts, respectively. Only few of the obtained results are statistically significant at the chosen α level (marked in bold). As expected, in most of the cases, statistical significance testing did not provide meaningful results ($\beta\text{-}1 << 80\%$). The only powerful results are marked in italics. Summarizing, we conclude that hypothesis ***H1 is valid***.

Table 4. Effort estimation accuracy

	CoBRA		OLS		Expert		
Obj.	***MMRE***	***Mean z***	***MMRE***	***Mean z***	***MMRE***	***Mean z***	***n***
O1	18.2%	97.9%	60.0%	44.1%	42.7%	57.3%	2
O2	25.4%	96.6%	34.0%	66.0%	37.1%	116.5%	4
O3	22.4%	101.%	32.1%	73.0%	36.0%	64.0%	3
O4	***24.1%***	98.6%	33.2%	66.8%	***44.4%***	55.6%	3
O5	39.6%	105.8%	46.3%	63.0%	72.2%	94.4%	3
O6	24.5%	93.1%	44.5%	55.5%	13.8%	88.8%	2

5.5.2. Hypothesis H2

The analysis of estimation precision (Table 5) suggests that, in principal, CoBRA® noticeably outperforms both expert judgment and OLS for all considered IV&V review objectives (a few exceptions are marked in gray). For example, it reduces RSD by 54% and 40%, on average compared to OLS and domain experts, respectively. Similar to accuracy, in most of the cases, statistical significance test-

ing did not provide meaningful results (β-1 << 80%). Summarizing, we conclude that hypothesis ***H2 is valid***.

Table 5. Effort estimation precision

	CoBRA		OLS		Expert		
Obj.	*Pred.25*	*RSD*	*Pred.25*	*RSD*	*Pred.25*	*RSD*	*n*
O1	80.0%	**26.3%**	20.0%	**74.0%**	00.0%	71.3%	2
O2	60.0%	10.7%	60.0%	17.6%	25.5%	12.6%	4
O3	60.0%	15.1%	60.0%	42.1%	33.3%	22.6%	3
O4	***50.0%***	09.9%	50.0%	20.6%	00.0%	***17.3%***	3
O5	00.0%	07.7%	33.3%	12.5%	33.3%	15.0%	3
O6	80.0%	***07.8%***	40.0%	***28.1%***	100.0%	01.9%	2

5.6. Threats to Validity

Several threats to the validity of the presented case study were identified. First, the very sparse project measurement data available prevented us from achieving sufficient power of performed statistical tests. Moreover, expert estimates used to compare CoBRA®'s performance were available only for some of the past projects considered in the study. Finally, the conclusions drawn in the study are limited to the specific context of IV&V reviews at JAMSS. Generalization of the study findings requires further replications.

6. Lessons Learned

The following practical lessons were learned while applying the CoBRA® method for estimation effort of IV&V:

(**LL1**) *Effort estimation scope*: Since IV&V activities differ depending on the objective of IV&V, the scope of effort estimation (the context for which an effort model is built) should be limited to a single IV&V objective. Total effort is the sum of effort over all objectives.

(**LL2**) *Size and complexity of review*: The complexity of a document under review should be considered as an effort driver beyond simple size measures, such as number of document pages.

(**LL3**) *Effort drivers*: Considering effort drivers other than size is a very important aspect of effort modeling. We experienced that a single factor may multiply effort by as much as 10 times (e.g., a complete lack of risk assessment already done by a software supplier may increase the effort of independent risk analysis by up to 20 times). Such an effect is impossible to investigate based only on historical size and effort data.

7. Summary

In this paper, we proposed adapting the CoBRA® software estimation method to predict the effort of independent verification and validation (IV&V). The method provides a potential solution to estimation problems in the context of IV&V. By integrating data- and expert-based estimation, CoBRA® requires minimal amounts of project measurement data and reduces the involvement of domain experts. As a result, it provides a reusable model that supports strategic project/process objectives, such as risk management for effort overrun for each IV&V objectives. At the same time, as reported by several empirical studies, it provides accurate and precise estimates.

When applied in the context of an example IV&V organization, CoBRA® proved to provide more reliable estimates than both the expert-based estimation currently applied and ordinary regression (OLS) - one of few data-intensive methods applicable in the IV&V context. It improved the accuracy and precision of estimates by 40%, on average. At the same time the method provided a transparent, context-specific effort model that supported IV&V practitioners in achieving project and process management objectives (e.g., negotiating project scope or improving the effectiveness of IV&V activities).

References

[1] B.W. Boehm, C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D. Refer, and Steece B. *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000.

[2] B.W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

[3] G. Boetticher, "An Assessment of Metric Contribution in the Construction of a Neural Network-Based Effort Estimator", *Proc. Int'l Workshop Soft Computing Applied to Soft. Eng*., 2001, pp. 59-65.

[4] S.D. Conte, H.E. Dunsmore, V.Y. Shen, *Software Engineering Metrics and Models*. The Benjamin-Cummings Publishing Company, Inc. 1986.

[5] R.N. Charette, "Why Software Fails [Software Failure]," *IEEE Spectrum*, vol. 32, no. 9, Sept. 2005, pp. 42-49.

[6] T. Foss, E. Stensrud, B. Kitchenham, I. Myrtveit, "*A simulation study of the model evaluation criterion MMRE*," IEEE Trans. Soft. Eng., vol. 29, no. 11, Nov. 2003, pp. 985-995.

[7] *IEEE Std. 1012-2004, IEEE Standard for Software Verification and Validation*, IEEE computer Society, June, 2005.

[8] N. Kohtake, A. Katoh, N. Ishihama, Y. Miyamoto, T. Kawasaki, M Katahira, "Software Independent Verification and Validation for Spacecraft" *Proc. IEEE JAXA Aerospace Conference*, 2008.

[9] M. Lother, R. Dumke, "Point Metrics. Comparison and Analysis," in *Current Trends in Software Measurement* (ed. R. Dumke, A. Abran), Shaker Publ., 2001, pp. 228-267.

[10] E. Mendes, "A Comparison of Techniques for Web Effort Estimation*," Proc. Int'l Symp. Emp. Soft. Eng. & Meas*., Madrid, Spain, 20-21 Sept., 2007, pp. 334-343.

[11] T. Menzies, J. Hihn, "Evidence-based Cost Estimation for Better Quality Software," *IEEE Software*, vol. 23, no. 4, 2006, pp. 4-6.

[12] M. Ruhe, R. Jeffery, I. Wieczorek, "Cost Estimation for Web Applications," *Proc. Int'l Conf. Soft. Eng*., 2003, pp. 285-294.

[13] D.J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedure*, (3rd ed.), Chapman & Hall/CRC; 2003.

[14] P. Sentas, L. Angelis, I. Stamelos, G.L. Bleris,. "Software Productivity and Effort Prediction with Ordinal Regression," J. Inf. & Soft. Tech., vol. 47, no. 1, Jan. 2005, pp. 17-29.

[15] Q. Song, M. Shepperd, M. Cartwright, C. Mair, "Software Defect Association Mining and Defect Correction Effort Prediction," *IEEE Trans. Soft. Eng*., vol. 32, no. 2, 2006, pp. 69-82.

[16] A. Trendowicz, "Software Effort Estimation – Overview of Current Industrial Practices and Existing Methods," Tech. Rep. 06.08/E, Fraunhofer IESE, Kaiserslautern, Germany, 2008.

[17] A. Trendowicz, J. Heidrich, J. Münch, Y. Ishigai, K. Yokoyama, N. Kikuchi, "Development of a Hybrid Cost Estimation Model in an Iterative Manner," *Proc. Int'l Conf. Soft. Eng.*, 2003.

Unified Basic Concepts for Process Capability Models

Clenio F. Salviano and **Adriana M. C. M. Figueiredo**
Centro de Pesquisas Renato Archer – CenPRA
Rod. D. Pedro I, km 143,6 – CEP 13069-901 - Campinas, SP, Brazil
Clenio.Salviano @ { cenpra.gov.br, gmail.com }, Adriana.Figueiredo @ cenpra.gov.br

Abstract
This article introduces a set of unified basic concepts for Process Capability Models for the unification, generalization and modeling of views of the structure and elements of relevant models, with more similar granularity of its leaf elements. This set is part of an ongoing research effort to evolve the current Software (and Systems) Process Improvement area towards a Model Driven Process Capability Engineering for Knowledge Working Intensive Organizations. The set is implemented in Eclipse Ecore. Its evaluation includes the modeling of a unified view of CMMI-DEV and ISO/IEC 15504-5 models as relevant examples.

Keywords
Software Process Improvement (SPI), Model-Driven Engineering (MDE), and Knowledge Working

1. INTRODUCTION

In a panel on Research Directions in Software Process Improvement (SPI), David Card pointed out that SPI "has become a driving force in the global software industry. … [however the majors SPI] approaches today are considered competitors. In reality they are all based on very similar concepts and techniques. The packaging obscures the underlying principles. Eliciting and refining underlying principles is the role of science" [1].

This article introduces a set of unified basic concepts as a proposal for the underlying principles of Process Capability Models that are a relevant part of SPI (now redefined as Software and System Process Improvement). SPI was established as a "driving force in the global software industry" around the development and successful usage of the Capability Maturity Model for Software (SW-CMM) [2]. Nowadays, the Capability Maturity Model Integration for Development (CMMI-DEV) [3], the successor of SW-CMM, and the ISO/IEC 15504-5 Exemplar Process Assessment Model for Software Engineering (ISO/IEC 15504-5) [4] are the dominant reference models for SPI. A more generic term – Process Capability Model – is proposed to mean all good practices reference models organized with the concept of Process Capability. Three more generic terms are also introduced: Process Capability Area, Process Capability Level and Process Capability Profile.

Process Capability Area is a set of related specific "what to do" good practices. Process Capability Level is a set of related generic "how good to do" good practices. Process Capability Profile is a model of a process, under the aspect of process capability, composed of pairs of a process capability area at a process capability level.

The article is organized as follows. This first section is an introduction to the article. The second section presents the research context and methodology. The third section defines goals. The fourth section introduces a class diagram for the unified basic concepts. The fifth section describes CMMI-DEV and ISO/IEC 15504-5 views. The sixth section provides evaluation against the goals. The seventh section presents related and future work. Finally the eighth section presents conclusions.

2. RESEARCH CONTEXT

The set of unified basic concepts introduced in this article is part of an ongoing research effort [5, 6, 7, 8] proposing a Model-Driven Process Capability Engineering for Knowledge Working Intensive Organizations (MDPEK) for the evolution of the current SPI area.

MDPEK is a Model Driven Engineering (MDE) [9, 10] for improving knowledge working (including software and system working) intensive organization, identifying and acting in relevant processes based on the concept of process capability, integrated with the organization strategy to better business results, driven by a Process Capability Profile defined with elements from one or more Process Capability Models [8]. The models are, for example, CMMI-DEV, ISO/IEC 15504-5, iCMM, eSCM-SP, OPM3, COBIT, ITIL, COMPETISOFT and MR-MPS and/or process capability views of different types of good practices models, such as ISO 9001, PMBOK, EFQM, SWEBOK and Agile Methodologies[1].

MDPEK focus on knowledge working processes, as defined first by Drucker [11]. In this sense, Process Capability Models not related with software or systems may also be used, as for example, the Process Capability Model for University Research Laboratories [12].

This research effort uses an industry-as-laboratory approach as proposed by Potts [13]. Potts argues that the traditional research-then-transfer approach is inadequate because it treats research and its application by industry as separate, sequential activities.

Following the industry-as-laboratory approach, a MDPEK exemplar methodology (named as PRO2PI for

[1] These fourteen models are well known. References for them are available elsewhere, for example, in Salviano [6, 8].

Process Capability Profile to Process Improvement) has been developed and applied in the industry together with the development and utilization of MDPEK and the set of unified basic concepts [5, 6, 7, 8].

The set of basic concepts introduced in this article is the result of the second phase of a project in this research effort. Phase 1 is a proposal and initial specification as described by Salviano [6]. Phase 2 is the development and implementation of a revised version and the modeling of CMMI-DEV and ISO/IEC 15504-5 models as examples.

3. RESEARCH GOALS AND METHODOLOGY

To guide the activities of the second phase, a main goal and five unfolded objective goals are defined. The main goal is that the unified basic concepts represent a useful proposal for the underling basic concepts of Process Capability Models. The first unfolded objective goal (Goal 1) is that the set of unified concepts generalize and unify the structure of CMMI-DEV and ISO/IEC 15504-5 models. The Goal G2 is that it has fewer, more generic and more flexible key basic concepts than the CMMI-DEV and ISO/IEC 15504-5 models.

The Goal G3 is that it supports the mapping and unification of the elements of CMMI-DEV and ISO/IEC 15504-5 models. The Goal G4 is that it supports a hierarchy of elements and represents the leaf elements with more similar granularity, than the correspondent elements of CMMI-DEV and ISO/IEC 15504-5 models.

Finally, the Goal G5 is that it supports the definition of process capability profiles with any combination among the elements of available models, reusing elements, preserving the original models and without the need to create new models.

4. *Geraes* CLASS DIAGRAM FOR THE UNIFIED BASIC CONCEPTS

The set of unified basic concepts for Process Capability Models is represented as the *Geraes* Class Diagram as illustrated in Figure 1 defined with TopCased [15], an Eclipse Modeling Framework [14] plugin to the definition of metamodels based on Ecore.

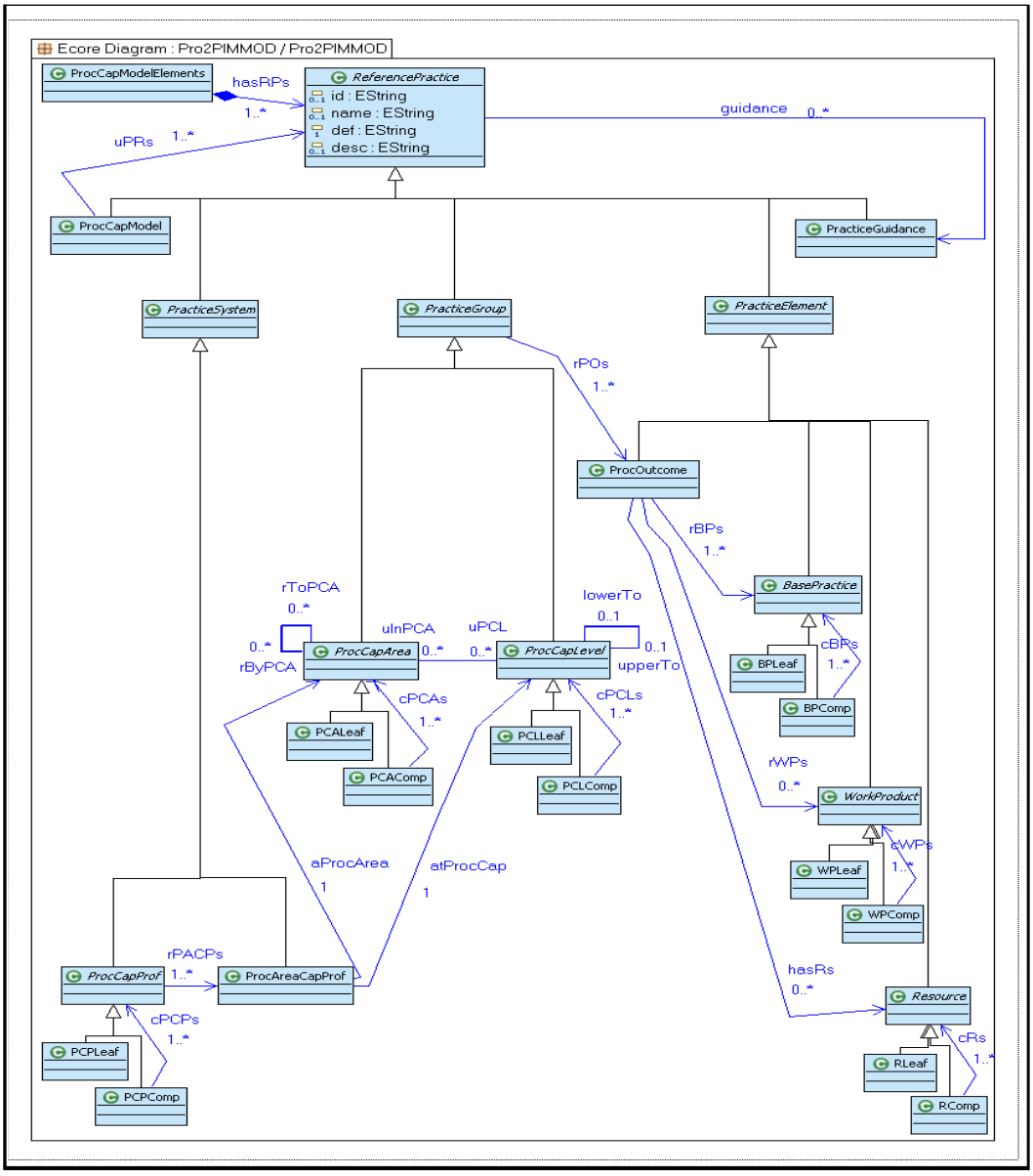


Figure 1 – *Geraes* Class Diagram in Eclipse Ecore

In Figure 1, for legibility's sake, the words Process, Capability and Profile are replaced by Proc, Cap and Prof.

An organization defines and uses a *ProcessCapabilityProfile* as a model of its current or future process. A *ProcessCapabilityProfile* is composed of one or more *ProcessAreaCapabilityProfiles*. Each *ProcessAreaCapabilityProfile* is a *ProcessCapabilityArea* at a *ProcessCapabilityLevel*. Each *ProcessCapabilityArea* and each *ProcessCapabilityLevel* is composed of one or more *ProcessOutcomes*. Each *ProcessOutcome* is composed of one or more *PracticeElements*.

A *ProcessCapabilityModel* is a collection of one or more *ReferencePractices*. *ReferencePractice* is an abstract super class for (with the exception of *ProcessCapabilityModelElements*) all Process Capability Model elements, including the *ProcessCapabilityModel* itself. There are four attributes for *ReferencePractice*: id (identification), name, def (definition) and desc (description). *ProcessCapabilityModelElements* is a collection of all *ReferencePractices*. A *ReferencePractice* may refer to one or more *PracticeGuidances*. *ProcessCapabilityModel* and *PracticeGuidance* are two concrete subclasses of *ReferencePractice*. There are also three more subclasses of *ReferencePractice*: the abstract classes *PracticeElement*, *PracticeGroup* and *PracticeSystem*.

PracticeElement has four subclasses: *ProcessOutcome*, *BasePractice*, *WorkProduct* and *Resource*. *PracticeGroup* has two subclasses: *ProcessCapabilityArea* and *ProcessCapabilityLevel*. *PracticeSystem* has two subclasses: *ProcessCapabilityProfile* and *ProcessAreaCapabilityProfile*.

ProcessAreaCapabilityProfile represents a *ProcessCapabilityArea* at a *ProcessCapabilityLevel*. *ProcessCapabilityProfile* is a collection of one or more *ProcessAreaCapabilityProfiles*. A *ProcessCapabilityProfile* is a collection of one or more *ProcessAreaCapabilityProfile*.

Each *ProcessCapabilityArea* and *ProcessCapabilityLevel* is a collection of one or more *PracticeElements*. A *PracticeElement* is a collection of one or more *ProcessOutcomes*. A *ProcessOutcome* is a collection of one or more *BasePractices*, zero or more *WorkProducts* and zero or more *Resources*. *ProcessOutcome*, *BasePractice*, *WorkProduct* and *Resource* are subclasses of *ProcessElement*.

The concrete class *ProcessAreaCapabilityProfile* is a connection between *PracticeSystem* and *PracticeGroup*. The concrete class *ProcessOutcome* is a connection between *PracticeGroup* and *PracticeElement*.

ProcessCapabilityArea, ProcessCapabilityLevel, ProcessOutcome, BasePractice and *ProcessCapabilityProfile* are modeled with two more subclasses each, using the Composer Design Pattern [16]. This design pattern addresses the need to compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. Each one of these classes is modeled as an abstract class with two concrete subclasses each: the first one models the composition relation (with the Comp suffix) and the second one models the leaf element (with the suffix Leaf). The composition and the leaf element use as a name the aggregation of the initials of the root element name (PCA, PCL, PO, PP, PCP and PCM) followed by the suffix Elem or Comp.

The class diagram in Figure 1 is named ***Geraes*** because its shape resembles the drawing in the cover of Milton Nascimento´s album named Geraes [17][2].

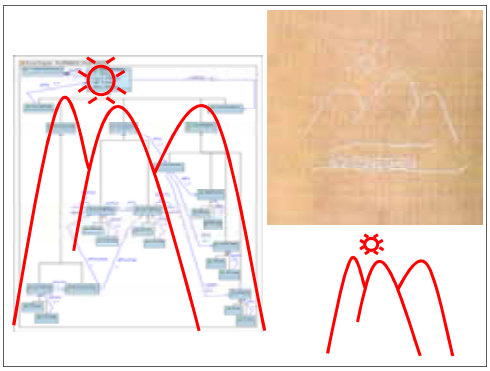


Figure 2 – *Geraes* Diagram and Geraes Album Cover

5. CMMI-DEV AND ISO/IEC 15504-5 VIEWS

This section is a partial unified view of the complete modeling of CMMI-DEV and ISO/IEC 15504-5 models that is described in a technical report [18].

A CMMI-DEV´s Process Area is modeled as a *ProcessCapabilityArea* composed of one, two or three lower level *ProcessCapabilityAreas*. Those lower levels *ProcessCapabilityAreas* correspond to the CMMI-DEV´s Specific Goals. There are twenty-two process areas and forty-eight specific goals in CMMI-DEV. The process areas are organized into four process area categories. Each process area category is modeled as a higher level *ProcessCapabilityArea*. Therefore these three CMMI-DEV concepts (process area category, process area and specific goal) are modeled as *ProcessCapabilityAreas* in three levels of composition.

The CMMI-DEV Project management process area category, for example, is modeled as a *ProcessCapabilityArea* composed of six lower levels *ProcessCapabilityAreas* (with ids "PP", "PMC", "SAM", "IPM", "RSKM" and "QPM"). The PP *ProcessCapabilityArea,* for example, is composed of three lower levels *ProcessCapabilityAreas* (with names "Establish Estimates", "Develop a Project Plan" and "Obtain Commitment to the Plan". Each one of these three lower levels *ProcessCapabilityAreas* is a *ProcessCapabilityAreaLeaf*, while each one of the two higher levels *ProcessCapabilityAreas* ("Project Management" and "Project Planning") is a *ProcessCapabilityAreaComp*.

[2] The idea of an icon is inspired in Favre on the S Mega-Pattern: "The shape of the S pattern has been chosen to directly mimic the layout of the piece of art *One and Three Chairs* from Kosuth" [9, p. 6].

The four original CMMI-DEV process area categories, twenty-two process areas and forty-eight specific goals are modeled as forty-eight *ProcessCapabilityAreaLeaf*s and twenty-two *ProcessCapabilityAreaComps* concrete classes. These twenty-two *ProcessCapabilityAreaComps* are the four process area category and the eighteen process areas with two or three specific goals. The four process areas with only one specific goal are modeled as *ProcessCapabilityAreaLeaf* because there is no difference between the original process area and its one specific goal.

A CMMI-DEV specific goal is a *ProcessCapabilityAreaLeaf* composed of *ProcessOutcomes*. There is no concept of outcome in CMMI-DEV. A specific goal is composed of specific practices. In order to model an outcome, each specific practice is modeled as an outcome and as a practice. The outcome is the practice rewritten with the passive voice. The specific practice "SP 1.1 Elicit Needs", for example, is modeled as a *BasePractice* (id "SP 1.1", name "Elicit Needs", and def "Elicit stakeholder needs, expectations, constraints, and interfaces for all phases of the product lifecycle") and as a *ProcessOutcome* (id "Out 1.1", name "Needs are elicited", and def "stakeholder needs, expectations, constraints, and interfaces for all phases of the product lifecycle are elicited").

An ISO/IEC 15504-5´s process is modeled as a *ProcessCapabilityArea*. There are forty-eight processes in ISO/IEC 15504-5 model that are organized in nine process groups. Each ISO/IEC 15504-5 process groups is modeled as a higher level *ProcessCapabilityArea*. The ISO/IEC 15504-5 process groups are further organized in three process categories. Each ISO/IEC 15504-5 process category is a higher level *ProcessCapabilityArea*. Therefore, these three ISO/IEC 15504-5 concepts (process category, process group and process) are modeled as *ProcessCapabilityAreas* in three levels of composition.

The ISO/IEC 15504-5 organizational life cycle process category, for example, is modeled as a *ProcessCapabilityArea* (name "Organizational"). This *ProcessCapabilityArea* is composed of four lower levels *ProcessCapabilityArea*s (with names "Management", "Process Improvement", "Resource and Infrastructure" and "Reuse"). The Management *ProcessCapabilityArea,* for example, is composed of six lower levels *ProcessCapabilityAreaLeafs*.

An ISO/IEC 15504-5 outcome and its correspondent base practices are a *ProcessOutcome* with *BasePractices*. Each base practice corresponds to one or more outcomes. When a base practice corresponds to more than one outcome, it is decomposed in a way that each element corresponds to only one outcome. The ISO/IEC 15504-5 Risk management process, for example, is a *ProcessCapabilityArea* with six *ProcessOutcomes*. The first *ProcessOutcome* is modeled with desc equal to "the scope of the risk management to be performed is determined". This first *ProcessOutcome* has only one *BasePractice* (id "MAN1.BP1", name "Establish risk management scope", and desc "Determine the scope of risk management to be performed").

A CMMI-DEV process capability level is a *ProcessCapabilityLevel* composed of one, two or ten lower level *ProcessCapabilityLevel*. Those lower levels *ProcessCapabilityLevel* are named Generic Goals in CMMI-DEV. There are six process capability levels in CMMI-DEV and seventeen Generic Goals.

An ISO/IEC 15504-5 process capability level is a *ProcessCapabilityLevel*, composed of zero, one or two lower level *ProcessCapabilityLevel*. Those lower levels *ProcessCapabilityLevel* are named Process Attribute in ISO/IEC 15504-5. There are six process capability levels and nine process attributes in ISO/IEC 15504-5.

The six original CMMI-DEV process capability levels and the six original ISO/IEC 15504-5 process capability levels are modeled as six *ProcessCapabilityLevels* composed of zero, one, two or three lower level *ProcessCapabilityLevels*. As both sets are based in the measurement framework of ISO/IEC 15504, they are similar. But they are not equivalent. Therefore, each set is modeled separated, but using a similar structure in order to facilitate further composition among them.

The ISO/IEC 15504-5 "Managed process" capability level and the CMMI-DEV "Managed process" capability level are modeled as *ProcessCapabilityLevel* (id "2" and name "Managed process"). This capability level is defined in ISO/IEC 15504-5 as composed of two process attributes (PA2.1 and PA2.2). The corresponded capability level in CMMI-DEV is defined as composed of ten Generic Practices (GP2.1 to GP2.10). The correspondent *ProcessCapabilityLevel* is modeled as composed of three lower levels *ProcessCapabilityLevels*. Table 1 describes these three lower levels *ProcessCapabilityLevels*, identified in column "Unified Leaf PCL". Some of the original Generic Practices and Outcomes are divided into two parts, identified as (p1) and (p2), in order to allow a correspondence to the other model.

Using the contents of Table 1 as a guide, the capability levels of the original CMMI-DEV, the original ISO/IEC 15504-5 and a unified view, can be modeled. They use a subset of the same fourteen *ProcessCapabilityLevelLeafs*. Unified 2.1.1, 2.1.6 and 2.3.4 exist in CMMI-DEV but not in ISO/IEC 15504-5. Unified 2.1.2, 2.2.1 and 2.2.2 exist in ISO/IEC 15504-5 but not in CMMI-DEV. A similar table is produced for capability level 3 (with also three *ProcessCapabilityLevelLeafs*), for capability level 4 and 5 (each one with two *ProcessCapabilityLevelLeafs*).

A CMMI-DEV staged representation is modeled with four *ProcessCapabilityProfiles*, each one representing a maturity level.

Table 1 - Example of Unified View

Generic Practices for CMMI-DEV Generic Goal 2		Unified Leaf PCL	Outcomes for ISO/IEC 15504-5 Process Attributes 2.1 and 2.2	
Id	**Name**	**Id**	**Id**	**Name**
GP 2.1	Establish Organizational Policy	2.1.1		
		2.1.2	PA 2.1 a)	Objectives for the performance are identified;
GP 2.2	Plan the Process	2.1.3	PA 2.1 b) (p1)	Performance of the process is planned;
GP 2.3	Provide Resources	2.1.4	PA 2.1 e)	Resources and information necessary are identified, (…);
GP 2.4	Assign Responsibility	2.1.5	PA 2.1 d)	Responsibilities and authorities are defined, (…);
GP 2.5	Train People	2.1.6		
		2.2.1	PA 2.2 a)	Requirements for the work products are defined;
		2.2.2	PA 2.2 b)	Requirements for documentation and control are defined;
GP 2.6	Manage Configurations	2.2.3	PA 2.2 c)	Work products are appropriately identified, (…)
GP 2.9	Objectively Evaluate Adherence	2.2.4	PA 2.2 d)	Work products are reviewed and adjusted as necessary ;
GP 2.8 (p1)	Monitor and Control the Process	2.3.1	PA 2.1 b) (p2)	Performance of the process is monitored;
GP 2.8 (p2)	Monitor and Control the Process	2.3.2	PA 2.1 c)	Performance of the process is adjusted to meet plans;
GP 2.7	Identify and Involve Relevant Stakeholders	2.3.3	PA 2.1 f)	Interfaces between the involved parties are managed.
GP 2.10	Review Status	2.3.4		

The abstract class *PracticeElement* and its concrete subclasses *ProcessOutcome*, *BasePractice*, *WorkProduct* and *Resources* represent the elements of the original specific and generic goals of CMMI-DEV, composed of specific and generic practices and sub-practices and typical work products, and the original outcomes, base practices, work products and resources from ISO/IEC 15504-5.

6. EVALUATION

As an evaluation, the achievements of the five objectives goals are commented. The achievement of Goal G1 is evidenced by the modeling of CMMI-DEV and ISO/IEC 15504-5 models. The achievement of Goal G2 is evidenced by the number of key concepts. Seven explicit key CMMI-DEV concepts (process area, specific goal, process area category, process capability level, generic goal, maturity level, capability profile) and one key implicit concept (process area capability profile) are modeled with four key concepts of the class diagram (*ProcessCapabilityArea*, *ProcessCapabilityLevel*, *ProcessAreaCapabilityProfile* and *ProcessCapabilityArea*). Four explicit key ISO/IEC 15504-5 concepts (process, process capability level, process capability profile and process attribute) and one key implicit concept (individual process capability profile) are modeled with the same four key concepts of the class diagram. For *PracticeElement*, four explicit key CMMI-DEV concepts (specific practice, generic practice, typical work product and sub practice) are modeled with four key concepts of the class diagram (*BasePractice* and *WorkProduct*). Six explicit key ISO/IEC 15504-5 concepts (outcome, base practice, work product, generic practice, generic resource and generic work product) are modeled with the four key concepts of the class diagram (*ProcessOutcome*, *BasePractice*, *WorkProduct* and *Resource*). As the same eight key concepts cover twelve CMMI-DEV concepts and eleven ISO/IEC 15504-5 concepts, this goal is considered satisfied.

The achievement of Goal G3 also is evidenced by the description of how the key elements of both models are modeled. The achievement of Goal G4 is evidenced by an analysis of what elements from CMMI-DEV and ISO/IEC 15504-5 are modeled as *ProcessCapabilityAreaLeaf* and *ProcessCapabilityLevelLeaf*. There are differences in terms of granularity among the original twenty-two process areas from CMMI-DEV, among the original forty-eight processes from ISO/IEC 15504-5, and between the two models as well. The Project Management process from ISO/IEC 15504-5 is similar to the set composed of two CMMI-DEV process areas: Project Management and Project Monitoring and Control. These two process areas have five specific goals. In order to make feasible the relationship among these two models, both, the ISO/IEC 15504-5 process and the pair of CMMI-DEV process area, are modeled as five *ProcessCapabilityAreaLeafs*. The forty-eight original ISO/IEC 15504-5 processes are covered by seventy-two *ProcessCapabilityArea*, and, the original twenty-two process areas of CMMI-DEV are covered by forty-eight *ProcessCapabilityArea*, with almost all of them already included in the seventy-two from ISO/IEC 15504-5.

The achievement of Goal G5 is evidenced by the class *ProcessCapabilityModelElements* as a collection of all elements. The reference to the elements is the only self-contained reference and allows the reuse of each element in any model.

7. RELATED AND FUTURE WORK

Mappings, comparisons and harmonization among concepts and elements from Process Capability Models in general, and from a CMMI-DEV model and ISO/IEC 15504-5 model in particular, have been done elsewhere, as for example, by Lepasaar et al. [19], Alexandre and

Habra [20] and Rout and Tuffley [21]. The iCMM and MR-MPS models combine elements from a CMMI model and ISO/IEC 15504-5 model. All of these related works address the goals G1 and G3, but they do not address the goals G2, G4 and G5.

Siviy and Kirwan [22] present an initial reasoning framework for harmonizing process improvement efforts when multiple improvement technologies and models are in use. *Geraes* is a specific proposal, based in process capability, for that reasoning framework.

As a future work, a phase 3 of the project is under way in which *Geraes* is under another revision in order to support the concepts of the other models listed in section 2. A preliminary analysis pointed out that most of these concepts are already supported. This revision is also a consolidation of the unified basic concepts as a complete formal specification and implementation Process Capability Profile Metamodel in order to define a consensual agreement on how elements of a process should be selected to produce a given Process Capability Profile. The concept of metamodel is used as proposed by Favre [9] and Bézivin [10].

8. CONCLUSION

This article proposes *Geraes* class diagram as a set of unified basic concepts for Process Capability Models, addressing the issue identified by Card [1]. The class diagram is evaluated against five defined goals, including the modeling of CMMI-DEV and ISO/IEC 15504-5 models. This proposal is part of an ongoing research effort to evolve the current SPI area towards MDPEK and PRO2PI. The class diagram helps the understanding the concepts of Process Capability Models independent of how a particular model implemented these concepts. This proposal is a relevant step towards a process capability profile metamodel that will support MDPEK and PRO2PI and therefore the consistent utilization of elements from multiple models to process improvement.

ACKNOWLEDGMENTS

The authors would like to thank all people from many projects and institutions and the anonymous reviewers for their comments and suggestions.

REFERENCES

1 Card, D. N.: Research Directions in Software Process Improvement. In 28th IEEE International COMPSAC Conference, pp. 238--239, Hong Kong, China (2004)

2 Paulk, M. C., Weber, C. W., Curtis, B. and Chrissis, M. B.: The Capability Maturity Model, Addison-Wesley, 441 pages (1994)

3 Chrissis, M. B., Konrad, M., Shrum, S.: CMMI: Guidelines for Process Integration and Product Improvement, 2nd Edition, Addison-Wesley (2007)

4 ISO/IEC, ISO/IEC 15504 - Information Technology - Process Assessment – Part 5: An exemplar Process Assessment Model (2006)

5 Salviano, C. F., Jino, M., Mendes, M. J.: Towards an ISO/IEC 15504-Based Process Capability Profile Methodology for Process Improvement (PRO2PI), in Proc. of The Fourth International SPICE Conference, Lisbon, Portugal, p. 77-84, April 28-29 (2004)

6 Salviano, C. F.: Uma proposta orientada a perfis de capacidade de processo para evolução da melhoria de processo de software (in Portuguese) (A proposal oriented by process capability profiles for the evolution of software process improvement), PhD thesis, FEEC Unicamp, Campinas, SP, Brazil (2006)

7 Salviano, C. F. and Jino, M.: Towards a {(Process Capability Profile)-Driven (Process Engineering)} as an Evolution of Software Process Improvement. In: EuroSPI Industrial Proc, Finland, p. 12.26-37 (2006)

8 Salviano, C. F.: Model-Driven Process Capability Engineering for Knowledge Working Intensive Organization, accepted for publication in The Eighth International SPICE Conference, Germany, May (2008)

9 Favre, J-M: Foundations of Model (Driven) (Reverse) Engineering: Models - Stories of The Fidus Papyrus and of The Solarus, www-adele.imag.fr/mda (2005)

10 Bézivin, J.: On The Unification Power of Models, in Software and System Modeling – SoSym, 4(2), p. 171-188 (2005)

11 Drucker, P.: Landmarks of Tomorrow - A Report on the New 'Post-Modern' World, Harper & Row, New York (1959)

12 Silva, J. L. da, Nabuco, O. F., Salviano, C. F., Reis, M. C., and Maciel Filho, R.: Strategic Management in University Research Laboratories, in Proc. Fourth International SPICE Conference, South Korea (2007)

13 Potts, C.: Software-Engineering Research Revised, IEEE Software, 10(5), pp. 19-28, Sep. (1998)

14 Eclipse Modeling Framework, at www.eclipse.org

15 TopCased plugin, http://www.topcased.org/

16 Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley (1994)

17 Nascimento, M.: Geraes Album, EMI-ODEON(1977)

18 Salviano, C. F. and Figueiredo, A. M., Unified Basic Concepts of Capability Model for CMMI-DEV and ISO/IEC 15504-5 Models, CenPRA TecRep., (2008)

19 Lepasaar M., Mäkinen T., and Varkoi T.: Structural comparison of SPICE and continuous CMMI, In the Proc. of SPICE Conference, pp. 223-234, Italy (2002)

20 Alexandre, S., and Habra, N. UML modeling of five process maturity models. Tech. Rep. LQL-2003-TR-02, CETIC-FUNDP, Charleroi-Namur (2003)

21 Rout, T. P., and Tuffley, A.: Harmonizing ISO/IEC 15504 and CMMI, in Software Process Improvement and Practice, 12(4), pp. 361-371 (2007)

22 Siviy, J. M. and Kirwan, P.: Process Improvement in Multimodel Environments: Past, Present, Future. Software Engineering Institute, Carnegie Mellon University, USA, slides presented at SEPG Conference (2008)

Systematic Approach to Risk Management in Software Projects through Process Tailoring

Lisandra M. Fontoura[2,3] and Roberto Tom Price[1]
[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre RS, Brazil 91.501-970
[2]Centro Universitário Franciscano (UNIFRA)
Santa Maria RS, Brazil 97010-032
[3]Universidade Regional Integrada do Alto Uruguai e Missões
Santiago RS, Brazil 97700-000
e-mail: lisandra@urisantiago.br, tomprice@inf.ufrgs.br

ABSTRACT
In order to be effective in the generation of high quality products, a software process must suit the application domain, the organization's unique features, the development teams and the specific characteristics of the project. On the other hand, software projects involve development risks, depending on the complexity of the project, the team expertise, the technology deployed and many other factors. In this paper, a systematic approach is proposed to manage risks in software development projects through process tailoring. This approach aims at adapting a process to a specific project, with the objective of minimizing its exposure to its identified risks. Goal/Question/Metric plans are defined to monitor risks. Some developed case studies are presented.

1. INTRODUCTION
Tailored software processes are required so that projects can adopt development methods, techniques and practices according to their specific needs. The Software Engineering Institute (SEI), through Capability Maturity Model Integration (CMMI), proposes that organizations define a standard process, common to all the organizations' projects, and tailor this process according to the specific characteristics of each process [11]. However, process tailoring is not a simple task; it requires knowledge and experience in the software process. While a process is tailored, the characteristics of the project and of the environment must be taken into account. Dependable software development [15] for instance, requires particular environment features, tasks, and support tools to support the formalization of specifications, to demonstrate program correction and to create software replication, among others. Planned development process models [11] highlight detailed identification of processes activities and tasks and are suited to projects where a system quality improvement is required to make the projects more manageable, and to forecast dates and costs. These process models guide the team as they develop the work [11]. Agile models highlight flexibility, being suitable to development of software with unstable requisites, by small teams of highly skilled developers [11]. Both agile and planned approaches have context-dependent shortcomings that, if are not addressed, can lead to project failures. The challenge is to balance the two approaches to take advantage of their strengths in a given situation while compensating for their weakness [2].

Keeping the processes descriptions is important because they allow reusing the organizations' knowledge and processes to be assessed and improved [14]. Reusing processes is a way of reusing experience and knowledge, making possible to create a collection of reusable processes, which can be inter-connected to instantiate new and more complex processes. To produce high quality software in a competitive way, on schedule and within costs previously estimated, it is also necessary to manage the risks involved in the software development of a specific project [5].

This paper proposes a systematic approach - Project Risk Management Approach (PRiMA), which aims at tailoring the organization's process, to be used in a specific project, inserting in it preventive and corrective actions for the prioritized risks of the project. The preventive actions to risks are described as process and organizational patterns, since those catch successful practices in software management and can be used in the elaboration or improvement of the software processes [4]. The organization's standard process and the organizational and process patterns related to the risks that they intend to prevent are stored in a knowledge base that must evolve with time based on the developers experience. This makes it possible to reuse a usable set of software process assets that improve process performance across the projects and provide a basis for cumulative, long-term benefits to the organization [14].

A Project Risk Management Approach Tool - (PRiMA-Tool) was developed to support the use of the systematic approach proposed. This tool allows that the organization's standard process to be tailored to meet the specific needs of

a project, inserting in the project process, actions to prevent and monitor risks. As the tailoring result, a website is created with the description of the process for the project.

The paper is outlined as it follows: Section 2 shortly introduces organizational and process patterns; Section 3 explains the systematic approach proposed to manage risks in software projects, called PRiMA; Section 4 describes PRiMA-Tool. In Section 5 some case studies are presented. Section 6 mentions some related work, and in Section 7 conclusions and future works are suggested.

2. ORGANIZATIONAL AND PROCESS PATTERNS

According to Coplien [4], successful software organizations use the same organizational and process patterns in the software development. These patterns are not found in less productive or not so successful organizations [4]. A pattern describes the essential part of a solution for a recurrent problem in a specific context [8]. Organizational and process patterns catch successful practices in the management of software development [4] and can be used to model a new organization and a new development process for a project, or also to improve an existent process. When put into practice together in an organized way, process patterns can be used to create software processes for the organization. Process patterns are considered blocks of reusable processes that can be used to tailor the software process to find the specific needs of the organization [4].

In this work, the organizational knowledge to prevent risks is described through rules that associate one or more risks to one or more patterns to which they propose a solution, preventing or minimizing the risks. The rules are associated with project contexts, where they work better. [7]. Rules have an accuracy factor, which measures the efficiency of patterns associated in risk control. A factor of 0 (zero) means the patterns does not help at all, while a factor of 10 means the patterns can be applied to completely eliminate the associated risks. The accuracy factor can be adjusted, by users, from real experiences of the use of the rule. The project context is identified in terms of three criteria, using an approach similar to Cockburn [3] and Boehm [2]:

- Defects Criticality: The possibility of loss associated with the occurrence of a defect. It ranges from loss of comfort (1) to the loss of many human lives (5);
- Team Size: The number of people involved is also an important factor considered by Cockburn [3]. The larger the project team, more intermediate documents must be produced to coordinate the work;
- Team Skill: Boehm [2] extends Alistair Cockburn´s classification of people, and uses it as an important factor to balance between agile and plan-driven methodologies. The classification ranges from: *people who are unable or are unwilling to collaborate* (level -1) to *people who can revise a method, breaking its rules to fit an unprecedented new situation* (level 3). The team skill is calculated according to the percentage of people classified in the levels of understanding.

The association of patterns to risks was based on the existing knowledge in published books and articles as [2][11][14][1][12] and in catalogues of existing patterns [4][8]. Figure 1 shows examples of risk association rules, such as the risk *ambiguously or imprecisely written individual requirements* to patterns that intend to prevent it in different project contexts. Different rules are associated to the same risk in different contexts. For example, **Rule 1** has as goal to prevent risk in projects with planned characteristics – large teams with medium abilities and software of low criticality [4][12], while **Rule 2** will be used to select patterns in dependable projects – high skill, medium teams and high criticality [4][15], and **Rule 3** will be used in projects with agile characteristics – high skill, small teams and medium criticality [1][8]. The above associations of risks to patterns are examples; the organization has to adjust and tune them to its reality and to elaborate new rules according to its needs.

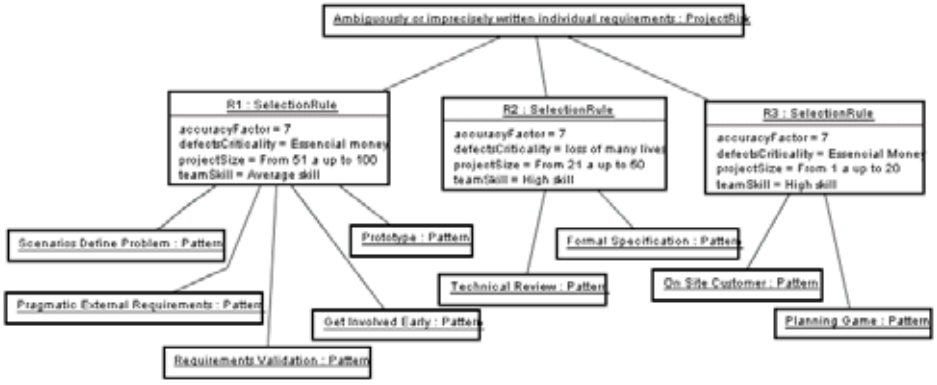


Figure 1 - Association of Patterns to Risks

An important characteristic of a process pattern is that it describes what must be done, but not exactly the details of how to do it [4]. It is necessary to identify the process elements required in the deployment of each pattern associated to the risks in a software process. Figure 2 shows process elements required to the deployment of the pattern *ScenariosDefineProblem*, as an example extracted from Coplien [4]. This association is based on the description of the pattern and on the practices described by models such as the CMMI [14], and by processes such as RUP [12], XP [1], PMBOK or by authors as Pressman [11], Sommerville [15], among others.

The activities, proposed by Rational Unified Process (RUP) [12], *Develop Vision*, *Find Actors and Use Case* and *Detail*

a Use Case are associated to the pattern *Scenarios Define Problem* (from Rule 1), as another example. Due to the readability of Figure 2 only the activity *Find Actors and Use Cases* was completely described, according to the description of RUP [12]. The process elements required to the insertion of this activity in a software process are highlighted in Figure 2. The activities, process elements associated to it, the patterns and the association rules are inserted in a knowledge basis.

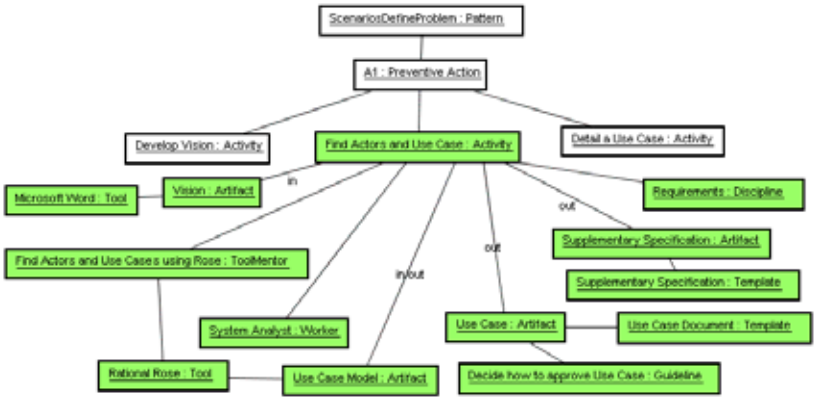


Figure 2 – Association of Pattern to Process Elements

3. PROJECT RISK MANAGEMENT APPROACH

The aim of the approach is to allow the elaboration of software development process for a specific project. This process has the objective of minimizing the project exposure to its identified and measured risks, according to the context of the project. Figure 3 shows the sequence of activities proposed by PRiMA.

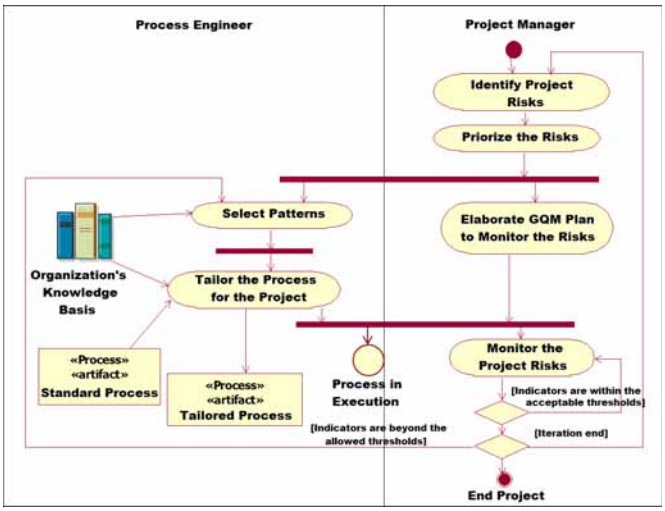


Figure 3 – Project Risk Management Approach

In the activity *Identify Project Risks,* the risks that may affect the project are identified. To help the project manager in this activity, the use of a common checklist of most frequently occurring risks is suggested [7].

To identify risks, the team members must be interviewed and group sessions must be organized with the team and with stakeholders involved in the project [5]. The process of identifying risks must not be limited to checklists proposed in the scientific literature. It is possible, for example, to create a list from the major problems that took place in the organizations past works, as suggested by DeMarco and Lister [5].

Considering that there can be a great number of identified risks for a project, each of a distinct nature and impact, in the activity *Priorize the Risks,* the risk exposure (RE) technique is employed to give priority to the risks that have more impact on the project and high or medium probability of taking place. Risk exposure, also called risk impact or risk factor, is the product of the probability of a non-satisfactory result to occur, and the loss associated to this non-satisfactory result. This work uses a 0 to 10 scale in order to measure the probability and the loss of each risk. Quantifying risks by risk exposure provides a relative priority order for all the identified risks. The project manager defines if all the risks identified will be handled, or just the ones with RE higher than a threshold. The cost to manage all the risks can be very high and risks with low probability of occurrence or low loss may not justify the cost to treat them [5].

The activity *Select Patterns* defines how to recover from the knowledge base, the patterns which aim to prevent the prioritized risks for a specific project [7]. In section 2 we described the proposed mechanism to select patterns from an organizational knowledge base.

The activity *Tailor the Process for the Project* describes how the process can be tailored to integrate the patterns selected to prevent the risks, resulting in the process defined for a project. The process tailoring is based on a process framework that integrates 1. the activities to be performed in all projects, 2. organizational patterns as preventive or corrective actions to risks, 3. GQM plans associated to risks. Each organization must define its own framework or tailor this to meet its specific needs. In the framework are described process tailoring guidelines and process configuration aiming to facilitate the tailoring task. The guidelines describe how to tailor process elements according to the size and formality of the project. Process configurations are pre-defined process models, aiming to meet typical projects or process improvement models, such as CMM. The framework structure is briefly described in section 3.1 of this paper.

During the project, it is necessary to monitor the risks to ensure that risks factors stay within the planned thresholds or to take some actions when they fall beyond the quantitative targets. In the activity *Elaborate the GQM Plan to Monitor the Risks,* the GQM Plan is defined. The GQM

Plan details the metrics that must be collected to answer the questions associated to the goals of the plan. The GQM goals must be formulated in the following way "Analyze the <*object of study*> aiming <*goal*> as regards to the <*focus*> in the point of view of <*point of view*> in the following context <*context*>". Table 1 shows, as an example, a GQM Plan to monitor the risk *ambiguously or imprecisely written individual requirements.*

Table 1. GQM Plan: Ambiguously or imprecisely written individual requirements risk

Risk: Ambiguously or imprecisely written individual requirements	
Goal: Analyze the project with the purpose of monitoring the requirements definition from the viewpoint of the development team.	
Question: Did the users validate the project requirements documents?	*Metric 1:* Percentage of Validated Requirements by the client RVRC = (amount of requirement documents validated/ total amount of requirement documents) *100

The risk monitoring is based on the metrics defined in the GQM Plan. The measures are stored in historical basis to be used in future estimates and in the project control. The metrics help the project manager to *Monitor the Project Risks*, providing visibility to their progress. The project manager must identify when corrective actions must be taken because the probability of occurrence of risk is growing. The probability of a risk to occur is defined by comparing the measurements done with pre-established thresholds. As results of the project risk monitoring, two situations may occur: 1. the measurements are within the acceptable thresholds, and a management action is not required; 2. the measurements are beyond the allowed thresholds, being necessary the execution of the activity *Select patterns* again to select other preventive actions to be inserted in the software project process.

At the end of any iteration, or periodically, the risks identified and prioritized for the project must be reviewed. In this review new risks may be identified, generating the need of tailoring the process again, and therefore new preventive actions can be included in the process to be used in the next iteration.

3.1 Framework

The basis of the tailoring task is a process framework, from which different processes can be instanced by the selection of process elements, previously defined by the organization. The framework is composed of a knowledge base, tailoring guidelines and process configuration.

The knowledge base comprises: software risks; instances of process elements (activities, roles, artifacts, disciplines, tools) used in the definition of software processes; activity diagrams describing the sequence of execution of activities by discipline; organizational and process patterns; rules of association patterns to risks; preventive actions describing the process elements needed to be deployed in patterns stored in the knowledge base; and goals, questions and metrics used to monitor risks.

Tailoring guidelines and process configurations are proposed to facilitate tailoring and definitions of the organization's process tasks. Process configurations are pre-defined process models, which include a set of process elements - they can be used as a starting point to define the organization's own process. These models were defined from well known processes or methodologies described in the literature. In the framework implemented with PRIMA-tool the following process configurations are suggested: simplified processes, as essential RUP [12], which describes a small set of elements to use the RUP in a project; complete, as RUP [12] and XP [1]; or extended to fulfill software improvement models, as RUP CMM Level 2 and RUP CMM Level 3. These process configurations are examples; every organization will define its process configurations according to its needs. SW-CMM and CMMI [14] consider that guidelines must be elaborated describing how to tailor the standard process of the organization to meet the specific needs of the project. These process tailoring guidelines are associated to the process elements and describe a list of alternatives to tailor a specific element. Process guidelines help the designer in tailoring processes, describing alternatives to tailor process elements according to the size and formality of the project. The guidelines are textual descriptions that describe details of how tailoring must be done to each process element [14].

4. PRIMA-TOOL

An experimental environment was developed composed of two tools: Pattern-Based Methodology Tailoring Tool (PMT-Tool) and Project Risk Management Approach Tool (PRiMA-Tool). PMT-Tool module was developed by Júlio Hartmann [7]. PMT-Tool is responsible for cataloguing the process patterns and associating them with the software risks by means of preventive rules, as well as selecting the pattern to prevent prioritized risks in a specific project. PRIMA-Tool module is responsible for the elaboration of the project software process, from organization's standard process tailoring, inserting in it the selected patterns to prevent the project risks and defining the Goal/Question/Metric Plans to manage the project risks. Having concluded the project process tailoring, Prima-Tool generates a website with the description of the defined process for the project to be consulted by developers, managers and process engineers. The tools are available for interested readers to play with at http://www.urisantiago.br:8080/prima/.

5. CASE STUDIES

Two case studies were carried out to validate the proposed framework. The case studies were made based on two software projects developed at a university, which will be called Y University. The projects show different characteristics, being one of the projects goal to develop a financial system in one of the Y University campuses (Fin$oft Project), while the other project goal is to develop

an academic system in a distributed way, by teams located at three different campuses of the same university (@cadSoft Project). Y University standard process is very simple, and it is based on a small subset of RUP activities, proposed by Essential RUP [12]. Table 2 shows in column 2, the activities that make the Y University standard process.

The team that will develop the Fin$oft software is composed of a project manager, two developers, two trainees and the Computing Center Coordinator, who is the Senior Manager of this project. The team is classified as a high-skilled team, as developers are experienced in the programming language employed and have already developed similar systems. The defined risks for the Fin$oft project are: *failure to manage end user expectations*, *misunderstanding the requirements*, *conflict between user departments*, *scope and goals are not clearly defined*, and *non-realistic schedule and budget*. Among the pattern list suggested by the PRiMA Approach, the patterns selected were: *EarlyAndRegularDeliverXP*, *PlanningGame*, *OnSiteCustomer*, *ConstantRefactoring*, *BuildPrototype*, *DocumentedSoftwareEstimate, SimpleDesign, SizeThe-Schedule and DocumentedConfigurationManagementPlan.*

Column 3 in table 2, shows the activities to be added to Y University's standard process (column 2) to form the specific process for the Fin$oft Project. The team that is developing Fin$oft financial software is small, with developers who are knowledgeable on the technology to be used in the project. The client is within the campus where the system will be developed, so the project configuration can use most of the agile methodologies. The main patterns suggested by PRiMA are those that aim to provide more agility to the organization standard process, based on RUP.

Another case study, @cadSoft academic software is being developed by three teams, totalizing 22 people: 3 project managers, 8 developers, 10 trainees and 1 senior manager, based on the Y University administrative area. The team is classified as average skilled, due to the large number of trainees and the fact that the team does not master the technology chosen for the system development. The defined risks for the @cadSoft project are: *lack of a methodology for the project*, *lack of required knowledge/skill in the project*, *misunderstanding the requirements*, *introduction of new technology*, *wrong development of functions of user interfaces*, *unfeasible design*, and *lack of top management commitment to the project*.

Table 2. Organization's standard process and Fin$oft's and @cadSoft's defined processes

Discipline	Organization's Standard Process	Fin$oft	@cadSoft
Requirements	Develop Requirements Management Plan	Write User Story Divide User Story Priorize User Story Develop a System Prototype Validate a System Prototype	Find Actors and Use Cases Structure the Use Case Model Detail a Use Case Review Requirements
Analysis and Design	Architectural Analysis Class Design Database Design	Write Tasks	Asses Viability of Arch. Proof-of-concept Construct Architectural Proof-of-concept Describe the Run-time Architecture Implement Innovative Idea Review the Architecture Review the Design Validate Innovative Idea
Implementation	Execute Developer Tests Implement Design Elements	Execute Unit Tests Implement Tasks Refactor Code	Execute Unit Tests Refactor Code Review Code
Test	Define Test Approach Define Test Details	Execute Acceptance Tests Write Acceptance Tests Write Unit Tests	Execute Acceptance Tests Write Acceptance Tests Write Unit Tests
Deployment	Create Deployment Unit Develop Support Materials		
Configuration and Change Management	Confirm Duplicate or Rejected CR Review Change Request Submit Change Request Update Change Request	Establish Change Control Process Write CM Plan	Establish Change Control Process Write CM Plan
Project Management	Compile Software Development Plan Develop Business Case Develop Iteration Plan Identify and Assess Risks Initiate Project Iteration Acceptance Review Lifecycle Milestone Review Prepare for Phase Close-out Prepare for Project Close-out Project Acceptance Review Report Status	Accept Task Classify by Risk Collect Metrics Define Iteration Scope Define Control Processes Define the Scheduler Define Velocity Develop Measurement Plan Elaborate Release Plan Estimate Task Estimate User Story Monitor Status Project Negotiate the Scheduler Team Negotiate with Customer	Assess Iteration Initiate Iteration Iteration Planning Review Plan Phases and Iterations Project Approval Review Project Planning Review Project Review Authority (PRA) Review
Environment	Select and Acquire Tools Set Up Tools Tailor the Process for the Project		Develop Development Case Review the Software Process for the Project
Training			Identify Training Execute Training

Among the suggested patterns listed to prevent risks, the following were selected: *ConstantRefactoring*, *SoftwareLifeCycleIsDefined*, *EarlyRegularDeliverRUP*, *ScenariosDefineProblem*, *ShunkWorks*, *ArchitectureTeam*, *ProjectProcessIsDefined*, *PeerReviews*, *ApprenticeShip*, *SeniorManagementReview* and *DocumentedConfiguration-ManagementPlan.* Column 4 shows the activities added to the Y University standard process (column 1) to form the specific process for the @cadSoft Project.

In the @cadSoft Academic System's process, the situation presented is opposed to that of the Fin$oft, because the team is distributed in different campuses, and the customer, in this case the University administrative area, is far from the development teams, the team is composed of many trainees, who usually stay in the team only for short periods. The most relevant patterns, suggested by PRiMA-Tool, are patterns which aim to provide more planning and documentation to the software process. Considering the difficulty in face to face communication in distributed teams, documents are generated so that the teams can communicate and keep informed.

PRiMA helps the process designer on performing the process tailoring. However, the process designer role is fundamental, and her careful empirical consideration and evaluation is necessary in order to tailor an adequate process for a given project. The tool is helpful because it suggests the most suitable patterns to the project, accordingly to the data which is recorded in its repository. It saves the time of the process designer to browse through dozens or even hundreds of patterns which are available.

6. RELATED WORKS

The proposed framework in this paper differs from other risk management approaches by proposing organizational and process patterns as risk preventive actions and for tailoring the organization's software process to prevent the risks identified for the project.

Gnatz et al [6] propose a framework to describe software processes. They do not consider quality standards or models during tailoring. Kiper and Feather [10] propose probabilistic models to manage risks, while this work uses risk exposition quantification. Probabilistic models are more complex and difficult to be used. Keshlaf and Hashim [9] propose a model for risk management and a tool, called SoftRisk, to support the process. It does not define how the risks are monitored and controlled during the software project. Roy [13] proposes a framework, called ProRisk, to manage risks in software projects. The framework requires a detailed analysis of the organization and the scope of the project to develop a group of risk factors and organize them in a way to reflect the different risk perspectives, making its use difficult.

7. CONCLUSIONS AND FUTURE WORKS

This article proposes a Project Risk Management Approach – PRiMA - which makes it possible to instantiate development processes tailored according to the identified and prioritized risks of the development project. The aim of tailoring is to elaborate a defined process to a project suitable to the project's context, taking advantages of agile methods, planned or hybrid, while preventing identified risks for the project. The approach presented in this paper is limited by the difficulties of validating the empirical knowledge of experienced process designers, software engineers and project managers, expressed as patterns and risk resolution rules. Nevertheless, it has a strong and original contribution to structure a systematic approach for capturing this knowledge and assisting process designers and project managers on leveraging it. The approach suggests that the use of risk analysis combined with organizational patterns is a promising way of overcoming the limitations of existing software process improvement frameworks. The information stored in the knowledge base can be updated and must improve with time and as the team gets more experience. Results of post-mortem analysis of projects can help in this task. Future works include the use of a workflow management system to create environments to support the execution of processes defined from PRiMA.

8. REFERENCES

[1] Beck, K. Embracing Change with Extreme Programming. IEEE Computer, Oct. 1999.

[2] Boehm, B; Turner, R. Using Risk to Balance Agile and Plan-Driven Methods. IEEE Computer, June 2003.

[3] Cockburn, A. Agile Selecting a Project's Methodology. IEEE Software, New York, v. 17. n. 4, p. 64-71, July 2000.

[4] Coplien, J. Sofware Patterns. Originally published by SIGS Books and Multimedia, 1996.

[5] DeMarco, T. Lister, T. Waltzing with Bears: Managing Risk on Software Projects. Dorset House Publishing Company, 2003.

[6] Gnatz, M. et al. The Living Software Development Process. Software Quality Professional, Milwaukee, v.5, n.3, June 2003.

[7] Hartmann, J.; Fontoura, L. M.; Price, R. T. Using Risk Analysis and Patterns to Tailor Software Processes. SBES, 2005.

[8] Hillside. Patterns Library. 2003. http://hillside.net/patterns/.

[9] Keshlaf, A. A.; Hashim, K. A Model and Prototype Tool to Manage Software Risks. Asia-Pacific Conference on Quality Software, 2000.

[10] Kiper, J. D.; Feather, M. S. A Risk-based Approach to Strategic Decision-Making for Software Development. 38th Hawaii International Conference on System Sciences, Australian, 2005.

[11] Pressman, R. Engenharia de Software. 6. ed. McGraw Hill, 2006.

[12] Rational Software Corporation. Rational Unified Process, version 2003.06.12. Cupertino, 2003.

[13] Roy, G. G. A Risk Management Framework for Software Engineering Practices. Australian Software Engineering Conference, 2004.

[14] Software Engineering Institute, Capability Maturity Model Integration, Version 1.1, Carnegie Mellon University, 2002.

[15] Sommerville, I. Engenharia de Software. Pearson Education, 2007.

PROCESS TAILORING BASED ON WELL-FORMEDNESS RULES

Eliana B. Pereira[1], Ricardo M. Bastos[1], Toacy C. Oliveira[2]

[1]*Faculty of Informatics, Pontifical University Catholic of Rio Grande do Sul, Brazil*
E-mail: {epereira,bastos}@inf.pucrs.br

[2]*School of Computer Science, University of Waterloo, Ont., Canada*
E-mail: toacy @csg.uwaterloo.ca

Abstract

Process tailoring consists on the manipulation of an existing software process to incorporate new elements or remove existing ones. However, it is essential to constrain tailoring activities to guarantee certain properties hold in the tailored process. Therefore it is necessary to define a properly mechanism for process tailoring capable to maintain the consistency and compliance of the resulting process. In this paper, we propose the use of a set of well-formedness rules required to ensure tailoring of standard software development process based on RUP process.

1 INTRODUCTION

The efforts to implement standard software development processes across large organizations have gradually increased in the last years. Because the product quality is associated to the process utilized in its construction [1] the goal is to adopt a well-defined process in software development.

The use of a standard software development process may allow the improvement of performance, predictability and reliability of the work processes; and the increase of productivity [2]. Furthermore, it also facilitates the implementation of the process capability maturity models such as ISO/IEC 15504 and Capability Maturity Model Integration (CMMI).

Nowadays, many "off-the-shelf" processes that facilitate the initial job of deciding the process elements have been suggested by the academy and industry. The most known one is the Rational Unified Process (RUP) [3] proposed by IBM Rational, which provides information regarding activities and tasks for software development and management.

However a "one process to fit all project needs" approach does not work in software development [4]. As each project is unique in terms of business domain, customer requirements and technology [2], there is a need to adapt the standard software development process to the requirements and the specific context of each project [5], [6].

The act of adjusting and/or particularizing the terms of a standard software development process to accommodate differences among projects is called tailoring [7]. It implies adding, deleting and/or modifying elements, and changing relationships. The result of tailoring activities is a bespoke software process, which is called project-specific software process.

Nevertheless, though recent studies have recognized the importance of process tailoring, it is important to stress the difficulties related to this task. Park et al. [5] and Xu [2] highlight as the lack of knowledge support may make tailoring a difficult task. Fitzgerald et al. [8] discuss some issues about how agile methods are used and tailored in practice, once there is not much knowledge about tailoring of agile methods. Yoon et al. [9] show the problems one might face in order to maintain the consistency between the tailored process and the standard software development process.

The main goal of this paper is to present a set of well-formedness rules based on a process metamodel to constrain tailoring activities as something that may contribute to some of the issues mentioned above. The well-formedness rules are based on the RUP metamodel [10]. We believe that these rules facilitate software development organizations to guarantee consistency on their tailored processes with their standard software development processes.

The paper is laid out as follows. Section 2 presents the relevant literature on process tailoring and describes how process tailoring is supported in the RUP. Section 3 shows the well-formedness rules. In Section 4, an example of usage is described followed by conclusions.

2 BACKGROUND

2.1 Process Tailoring

Nowadays, it is widely accepted that a standard software development process should be tailored to fit the needs of the projects context [11]. Important capability maturity models such as ISO/IEC 15504 and CMMI recommend

[1] *Study developed by the Research Group in Software Quality of the PDTI, financed by Dell Computers of Brazil Ltd. with resources of Law 8.248/91.*

tailoring standards before applying them to a specific project. For instance, CMMI, a current version of the SW-CMM, assumes that the best way to support the variation of the processes is to provide tailoring guidelines associated with the standard software development process to answer specific needs of the organization projects. These guidelines should specify which elements can be modified from standard software development process as well as in which circumstances this occurs. ISO/IEC 15504, on the other hand defines that organizations must have a strategy for tailoring projects assets. It implies the establishment and maintenance of a standard set of processes and strategies for tailoring project's needs. The processes should indicate applicability and expected performance, and identify detailed tasks, activities and associated artifacts.

Software development processes such as RUP, XP and OPEN also consider process tailoring. They provide some guidelines to orient the deletion of the non-applicable elements, addition of required elements, and modification of existing elements for a particular project.

Process tailoring is also not new in the literature [12] and it has recently become a target of attention to the software engineering community. Some researchers have been focusing their attention on project characteristics that influence tailoring decisions [13], [14]. The same issue has also been studied or understood as a "contingency factor" [15], [16]. Other authors have involved the knowledge management on process tailoring [2], [17].

The main interest of this paper is to guarantee the consistency between the tailored process and the standard software development process. A review of the literature though shows few studies addressed to this aspect. Yoon et al. [9] have proposed a systematic method to formalize a standard software development process, process tailoring and to verify the tailored process. The authors describe a set of tailoring operations to preserve the dependency relationships among the elements of the standard software development process. Although the authors have considered the conformity between the standard software development process and tailored process and suggested some operations for this purpose, their study have not considered several mandatory software process elements such as roles, tools, phases, etc.

Welzel et al. [18] have developed a method to process tailoring (ProcePT) based on a process model called GV-Model. The process model is formed by activities and artifacts that must be tailored in the project context. GV-Model provides a set of 90 conditions to hold the deletion of these activities and artifacts. The authors have also developed a tool to support the GV-Model in PROLOG. However, the limitations of this research are the same found in [9]. Additionally, the artifact deletion is the only tailoring operation considered.

Although some studies have tried to cover the standard compliance there is the need for additional research on the area, as it can be seen in a recent systematic review on process tailoring [6].

2.2 RUP and Process Tailoring

RUP is a software engineering process that can be adapted for a very large class of software systems, different apllication areas, different types of organizations and different project sizes. The act of "adapting the process" is what RUP calls by process tailoring. Bearing in mind that two projects within the same organization may also be different, RUP indicates the use of process tailoring for each software project. The result of this tailoring is part of what RUP calls a development case. A development case describes all activities, roles, artifacts, and templates that must be used in a software project.

To support process tailoring RUP provides a specific discipline called environment. The environment discipline focuses on the necessary activities to configure the process for a project. Besides, guidelines and white papers on tailoring RUP for different types of projects and domains are also available. The problem related to the guidelines and the environment discipline is their informallity, that is, no reference about RUP metamodel can be found on them. As a consequence it is not possible to verify the consistency of the tailored processes.

In this sense, we verify the need of a suitable specification of tailoring rules based on the process metamodel. Without these rules, it is impossible to guarantee the integrity of the relationships among the elements that compose the metamodel according to the standard software development process. Moreover, considering the complexity and the amount of elements that compose RUP, the process tailoring activities is impracticable without an automated support. In order to achieve that, it is necessary to specify a set of rules compliant to the process metamodel.

3 WELL-FORMEDNESS RULES TO RUP PROCESS TAILORING

The approach of this research to tailor RUP process is composed by a set of well-formedness rules to process tailoring. The rules lead to some tailoring operations that preserve dependency relationships among the elements that compose RUP process. These rules are based on an extension of original RUP metamodel presented in [19], which captures the elements and relationships required on tailoring activities. The extended metamodel is shown in Figure 1. The included classes in this metamodel from the original RUP metamodel are three: Task, Sub-artifact and Optional. And, the new relationships are: the self relationships of the Workflow Detail, Activity, Task and Sub-artifact classes and the relationship referred to as *modifies* created between the Activity and Sub-artifact classes.

The operations aimed by the well-formedness rules are to delete or add elements to RUP process. It has been developed rules for elements such as Disciplines, Workflow

Details, Activities, Artifacts and Sub-artifacts. It is important to note as mentioned above that sub-artifacts do not exist in the original RUP metamodel. So, it has been defined a new class in the proposed metamodel to this element, once it allows the representation of the parts of an artifact. The concept of sub-artifacts is required to the process tailoring since not all artifacts are generated by only one activity. Sometimes parts of an artifact can become from distinct activities. Thus, sub-artifacts are necessary in order to allow the deletion of an activity which is responsible for producing a sub-artifact no longer demanded for the software process. As a consequence, it is possible to select which parts of an artifact are necessary for a specific project during the process tailoring.

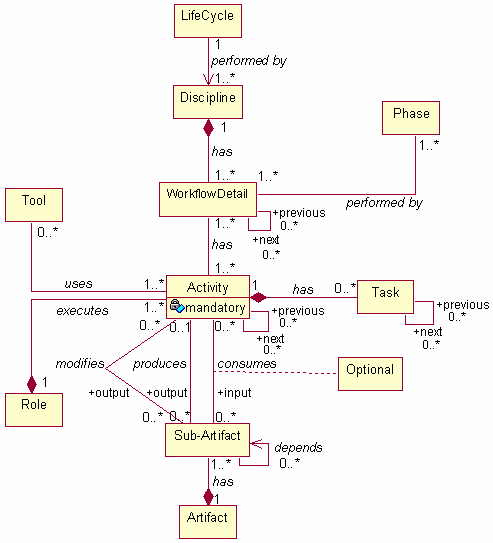


Figure 1 - The Extended RUP Metamodel

3.1 Tailoring Operation: Addition

As the UML multiplicity allows to refer to the number of objects in one class that can be related to one object in the related class and can be used on both sides of a relationship, it is considered that the well-formedness rules to addition operation are all expressed in the proposed metamodel. Thus, in order to add an element, the process engineer must respect the relationships of the added element only with other process elements. Details of the well-formedness rules to addition operations are provided bellow. Here, due to space constraints, we have decided not to illustrate the rules with tables as done in the following subsection.

Activity Addition: Adding activities instances implies in at least one relationship with the workflow detail element to indicate where the activity will be executed. Moreover, a responsible role for the activity execution and the related activities (precedent and subsequent ones) of the new activity must be defined. If necessary, the process engineer may associate tools to the new activity and create tasks to detail its execution. As activities have a clear purpose, usually expressed in terms of creating and/or updating artifacts, the process engineer may also associate the artifacts used by the new activity. Here, the process engineer uses the Sub-artifact class and the relationships referred to as *consumes*, *produces* and *modifies* to indicate which specific parts of an artifact will be consumed – in this case those mandatory or optional –, produced and/or updated by the new activity.

Workflow Detail Addition: When a workflow detail is added, at least one relationship with activity, discipline and phase instances must be created. In addition, related workflow details (precedent and subsequent ones) must be defined.

Sub-artifact Addition: When a process engineer wishes to produce additional information for a specific project he/she may add new sub-artifacts, which will be part of an artifact. In this tailoring operation, the process engineer must define at least one responsible role and optionally other roles that will update the new sub-artifact. The relationships referred to as *consumes*, *produces* and *modifies* may be used to associate the new sub-artifacts to different activities. Moreover, if the new sub-artifact has some kind of dependency relation to other sub-artifacts it must be created by the process engineer, who is also responsible for creating a relationship between the sub-artifact and an artifact.

Artifact Addition: Since we define an artifact as a set of sub-artifacts, the artifact addition operation implies in sub-artifacts addition. Thus, when an artifact is added at least one sub-artifact also must be included. However, it is known that when a process engineer applies the standard process to a small-size project or rapid application development, he/she may wish to manage process in higher-level to reduce workload and not to split an artifact. In this case, it is advisable to create for each artifact one sub-artifact with the same name of the artifact only. In ProTTo (our prototype cited in Section 4) it was implemented in the artifacts inclusion funcionality an option for process engineers to choose whether the included artifact need to be split or not. Thus, when an artifact is not split ProTTo automatically creates one sub-artifact with the same name of the included artifact.

Discipline Addition: Discipline addition is not supported since RUP disciplines cover all areas about software development.

3.2 Tailoring Operation: Deletion

Since RUP covers a large variety of development software, deletion operations are very common on process tailoring. For some of these operations, well-formedness rules are guaranteed in the present extended metamodel through UML relationships and its multiplicities. For instance, the composition relationships of the Discipline, Activity and Artifact classes, which define that the deletion of any of these elements implies in the deletion of its parts. However, many of the well-formedness rules can not be expressed using UML class model. So, we explain in a systematic way each deletion operation proposed in the present

approach. Bellow, all of them are illustrated with tables, since we may not express all well-formedness rules through the present extended metamodel as mentioned before. The tables show all affected elements by each tailoring operation and present a specific rule associated to each of them. Addiotionally, each rule contains a numeration to facilitate its identification.

Activity Deletion: Deleting activity istances of a standard software development process must be used only for optional activities (see the mandatory attribute of the Activity class in the Figure 1). In this operation, all activity tasks are deleted, since activities can be composed by tasks (see rule #9 of the Table 1). Moreover, related activities (precedent and subsequent ones) have to be connected in order to redefine the process workflow (see rule #1 of the Table 1). If a deleted activity produces sub-artifacts, it will be necessary to remove them from the activity (see rule #6 of the Table 1). In this case, if a removed sub-artifact has a dependency relationship with other sub-artifacts or it is mandatory consumed by other activities, other elements will have to be deleted (see rule #2 and #6 of the Table 1). Finally, all the associations of the deleted activity with the Role, Workflow Detail and Sub-artifact class have to be eliminated (see rule #3, #4, #5, #7 and #8 of the Table 1).

Workflow Detail Deletion: When a process engineer does not want to perform a group of related activities in a specific discipline he/she can delete workflow details. To do so, he/she must check whether the phases, activities and discipline related to the deleted workflow detail keep at least one relationship with another workflow detail. The phases, activities and disciplines that do not have other relationships also have to be deleted (see rule #1, #2 and #3 of the Table 2). It will also be needed to connect related workflow details (precedent and subsequent ones) (see rule #4 of the Table 2).

Table 1 - Well-Formedness Rules to Activity Deletion

Activity	#1	Related Activities (precedent and subsequent ones) must be connected.
	#2	Activities that consume the sub-artifacts (only the not optional ones) deleted by the activity must be eliminated.
Tool	#3	All the associations among the activity and tools must be eliminated.
Workflow Detail	#4	All the associations among the activity and workflow details must be eliminated.
Role	#5	All the associations among the activity and roles must be eliminated.
Sub-artifact	#6	Sub-artifacts produced by the activity and its dependent sub-artifacts must be deleted (this can imply in deletion of the other activities).
	#7	Sub-artifacts consumed by the activity must have its relationships eliminated.
	#8	Sub-artifacts modified by the activity must have its relationships eliminated.
Task	#9	All tasks of the activity must be deleted.

Table 2 - Well-Formedness Rules to Workflow Detail Deletion

Activity	#1	All relationships with activities must be deleted. If some activities no longer have relationship with other workflow details, they also must be deleted.
Discipline	#2	Workflow detail must be deleted from the discipline. If the discipline no longer has workflow details, it also must be deleted.
Phase	#3	All relationships with phases must be deleted. If some phases no longer have relationship with other workflow details they also must be deleted.
Workflow Detail	#4	Related workflow details (precedent and subsequent ones) must be connected.

Artifact Deletion: When an artifact is deleted its sub-artifacts must be deleted, too (see rule #1 of the Table 3). However, it will usually lead to other deletion operations (e.g. artifact deletion implies in executing deletion operation for all sub-artifacts of the deleted artifact).

Table 3 - Well-Formedness Rules to Artifact Deletion

Sub-artifact	#1	All sub-artifacts of the artifact must be deleted.

Sub Sub-artifact Deletion: Sub-artifact deletion is used to produce an artifact with different formats and levels of formality. Thus, when a process engineer wants to produce a less formal artifact he/she may delete pieces of this artifact, what can make it simpler. In order to delete a sub-artifact, a process engineer has to check if it is produced on software process or not. If so, he/she also has to delete its production activity (see rule #2 of the Table 4). Additionally, the process engineer must detele the sub-artifact of the artifact which it is associated and eliminate its associations with the Role class (see rule #1 and #4 of the Table 4).

Once the deleted sub-artifact is used by other activities, the process engineer must check whether it is mandatory or not. The activities where it is mandatory must be deleted (see rule #3 of the Table 4). Similarly, the process engineer has to check if the deleted sub-artifact has dependency relationships with other sub-artifacts, since these sub-artifacts also have to be deleted (see rule #5 of the Table 4).

Discipline Deletion: Deleting disciplines is implemented in the present approach following its use in RUP process, once some disciplines can be optionally performed (as Business Modeling, for example). In our approach, when disciplines are deleted the process engineer has to delete its workflow details (see rule #2 of the Table 5). Additionally, he/she has to eliminate the associations of the deleted discipline with the LifeCycle class (see rule #1 of the Table 5).

Table 4 - Well-Formedness Rules to Sub-Artifact Deletion

Artifact	#1	Sub-artifact must be deleted from the artifact. If the artifact no longer has sub-artifacts, it

		also must be deleted.
Activity	#2	The activity that produces the sub-artifact must be deleted.
	#3	The deleted sub-artifact must be disconnected of the activities. If some of these activities depend on the artifact to be performed they also must be deleted.
Role	#4	The association between the sub-artifact and roles must be deleted.
Sub-artifact	#5	The dependent sub-artifacts of the sub- artifact also must be deleted.

Table 5- Well-Formedness Rules to Discipline Deletion

LifeCycle	#1	The discipline must be deleted from the lifecycle. If the lifecycle no longer has disciplines, it also must be deleted.
Workflow Detail	#2	All workflow details of the discipline must be deleted.

4 EXAMPLE OF USAGE

We have applied the well-formedness rules to tailor RUP process in ProTTo – the tool prototype based on the proposed metamodel and on well-formedness rules to process tailoring [20]. We have considered the *Requirements* and *Analysis & Design* disciplines of RUP process as the standard software development process and we have tailored them by deleting some activities considered as optional on RUP process. Such activities are listed in Table 6.

Table 6 – Deleted Activities on Process Tailoring

Requirement Discipline Activities
Develop Requirements Management Plan (Deleted)
Find Requirement Attributes (Deleted)
Analysis and Project Discipline Activities
Perform Architectural Synthesis (Deleted)

In each deletion operation the side effects of the remained process elements were analysed, thanks to the impact analysis functionality available in ProTTo. This functionality allows any deletion operation, made in the process, to be propagated through out the various related elements. Thus, the affected elements are indicated by alert icones before the deletion is executed. Figure 2 illustrates the interface used in ProTTo to process tailoring. In this figure number 1 indicates the part of the interface used to perform the tailoring operations. **N**umber 2 and number 3 illustrate how a standard software development process is viewed in ProTTo. It is also possible to see an example of impact analysis in the figure in Number 2 and 3.

The first deleted activity in this example was the *Develop Requirements Management Plan*. This activity was performed in the standard software development process in a unique workflow detail called *Analyse the Problem*. Some partial results of the impact analysis can be found in Table 7. It shows tailoring operation led to the additional deletion of 2 artifacts, 8 sub-artifacts, 1 activity and 8 tasks. Moreover, the impact analysis shows that some relationships had also to be deleted, such as relationships among the deleted sub-artifacts and the activities that had consumed or modified them. Additionally, another impact caused by the tailoring operation, not shown in Table 7, is a process workflow change, that is, some deleted activities (*Develop Requirements Management Plan* and *Find Requirement Attributes*) were removed from the parallelism structures they had been configurated to work parallely with other activities in the workflow details in *Requirements* discipline.

We have also deleted the *Perform Architectural Synthesis* activity in the *Perform Architectural Synthesis* workflow detail. For this operation, the results of the impact analysis are shown in Table 8. Note that besides the exclusion of the *Perform Architectural Synthesis* activity other elements had also to be deleted. *In this case,* the tailoring operation led to the additional deletion of 1 artifact, 4 sub-artifacts and 5 tasks. Here, the process workflow had been changed for a second time in order to arrange the sequence of the activities execution in *Perform Architectural Synthesis* workflow detail.

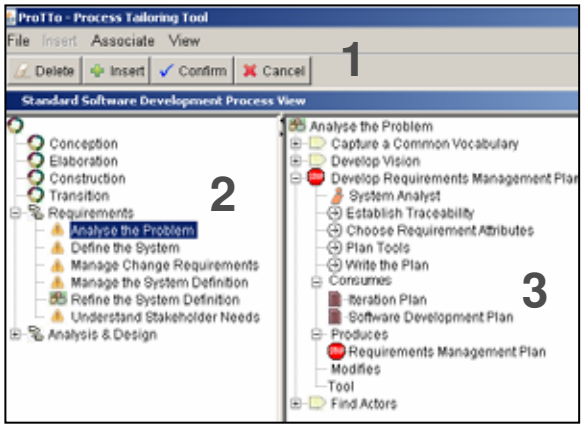


Figure 2 - Interface in ProTTo to Process Tailoring

5 CONCLUSIONS

In this paper, we have proposed capable well-formedness rules to lead the process tailoring. The main contribution of these rules is to guarantee the consistency between the tailored software process and the standard software development process in order to avoid tailoring unconformities. *In this paper, t*he focus has been on RUP process. However, we consider it possible to use this approach in other processes such as XP and OPEN, once these processes can also be tailored to specific projects. In this case, well-formedness rules must be adjusted considering the metamodel of each process.

REFERENCES

[1] Fuggetta, A., 2000. Software Process: A Roadmap. In: *Future of Software engineering Limerick Ireland, ACM.*

Table 7 - Partial Results of the Impact Analysis to Delete the Develop Requirements Management Plan Activity

Element	Instance	Reason	Well-Formedness Rules
Sub-Artifact	Sub-artifacts of the Requirement Management Plan Artifact	Sub-artifacts produced by the Develop Requirement Management Plan activity	Activity Deletion Operation Rule #6
	Sub-artifacts of the Requirement Attributes Artifact	Dependent sub-artifacts of the Requirement Management Plan artifact	Activity Deletion Operation Rule #6
Artifact	Requirement Management Plan	Artifact produced by the Develop Requirement Management Plan activity	Activity Deletion Operation Rule #6
	Requirement Attributes	Artifact produced by the Find Requirements Attributes activity	Activity Deletion Operation Rule #6
Activity	Find Requirements Attributes	Activity produces the sub-artifacts of the Requirement Attributes artifact that depend of the Requirement Management Plan artifact	Activity Deletion Operation Rule #6
Task	Tasks of the Develop Requirement Management Plan Activity	All tasks of the deleted activities must also be deleted.	Activity Deletion Operation Rule #9
	Tasks of the Find Requirements Attributes Activity		

Table 8 - Partial Results of the Impact Analysis to Delete the Perform Architectural Synthesis Activity

Element	Instance	Reason	Well-Formedness Rules
Sub-Artifact	Sub-artifacts of the Software Architecture Document Artifact	Sub-artifacts produced by the Perform Architectural Synthesis activity	Activity Deletion Operation Rule #6
	Sub-artifacts of the Deployment Model Artifact	Sub-artifacts produced by the Perform Architectural Synthesis activity	Activity Deletion Operation Rule #6
Artifact	Deployment Model	Artifact produced by the Perform Architectural Synthesis activity	Activity Deletion Operation Rule #6
Task	Tasks of the Perform Architectural Synthesis activity	All tasks of the deleted activities must also be deleted.	Activity Deletion Operation Rule #9

[2] Xu, P., 2005. Knowledge Support in Software Process Tailoring. In: *Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS).*

[3] Kruchten, P., 2000. *The Rational Unified Process: An Introduction*. Upper Saddle River, NJ: Addison-Wesley.

[4] Lindvall, M., Rus, I., 2000. Process Diversity in Software Development. In: *Institute of Electrical and Electronic Engineers - IEEE*.

[5] Park, S., Na, H., Park, S., Sugumaran, V., 2005. A Semi-Automated Filtering Technique for Software Process Tailoring Using Neural Network. In: *Expert Systems with Applications Journal*.

[6] Pedreira O., Piatiini M., Luaces M. R., Brisaboa N. R., 2007. A Systematic Review of Software Process Tailoring. In: *ACM SIGSOFT Software Engineering Notes*, Volume 32.

[7] Ginsberg, M. P., Quinn, L. H., 1995. ProcessTailoring and the Software Capability Maturity Model. *Technical Report*.

[8] Fitzgerald, B., Hartnett, G., Conboy, K., 2006. Customizing Agile Methods to Software Practices at Intel Shannon. In: *European Journal of Information Systems.*

[9] Yoon, I., Min, S., Bae, D., 2001. Tailoring and Verifying Software Process. In: *Institute of Electrical and Electronic Engineers - IEEE*.

[10] Bencomo, A., 2005. *Extending the RUP, Part 1*, viewed Dezember 02, 2007, < http://www.ibm.com/developerworks /rational/library/05/323_extrup1/index.html>.

[11] Fitzgerald B., Russo N. L., O'kane T., 2003. Software Development Method Tailoring at Motorola, In: *Communications of the ACM*.

[12] Basili, V. R., Rombach, H. D., 1987. Tailoring the software process to project goals and environments. In: *Proceedings of the 9th International Conference on Software Engineering Software*.

[13] Cockburn, A., 2000. Selecting a Project's Methodology. In: *Institute of Electrical and Electronic Engineers - IEEE*, July/August.

[14] Machado, L. F. C., 2000. *Modelo para Definição de Processos de Software na Estação TABA*. Master' Thesis. COPPE/UFRJ, Brazil.

[15] Rolland, C., Prakash, N., Benjamen, A., 1999. *Multi-Model View of Process Modeling Requirement Engineering*. Berlin: Springer-Verlag.

[16] Kraiem, N., Bourguiba, I. Selmi, S., 2000. Situational Method for Information System Project. In: *International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet*.

[17] Xu, P., Ramesh, B., 2003. A Tool for the Capture and Use of Process Knowledge in Process Tailoring. In: *Proceedings of the 36th Hawaii International Conference on System Sciences.*

[18] Welzel, D.; Hausen, H. L., Schmidt W., 1995. Tailoring and Conformance Testing of Software Processes. In: *IEEE*.

[19] Pereira, E. B., Bastos, R. M. e Oliveira T. C., 2007. A Systematic Approach to Process Tailoring, In: *International Conference on Systems Engineering and Modeling - ICSEM, Haifa, Israel.*

[20] Pereira, E. B., 2005. *Uma Proposta para Adaptação de Processos de Desenvolvimento de Software Baseados no RUP*, Master' Thesis. Faculty of Informatics. Pontifical Catholic University of Rio Grande do Sul – PUCRS.

Non-invasive software process data collection for expert identification

Andrea Janes, Alberto Sillitti, Giancarlo Succi
Free University of Bolzano/Bozen, Center for Applied Software Engineering
{ajanes, asillitti, gsucci}@unibz.it

Abstract

Software companies depend heavily on knowledgeable employees. Competence and skills management are essential instruments to understand how to employ the available skills in an optimal way.

Unfortunately, implementing knowledge management strategies like competence and skills management is challenging because resources, time and effort are required before benefits become visible.

This paper shows an approach to collect non-invasively (i.e., without requiring any effort by developers) data about "who" is working on "what" during software production.

We present two examples to show how to answer three questions: "who is the expert of a specific part of the code?", "who should do pair programming with whom?", and "what knowledge gap arises if a specific developer leaves?".

1. Introduction

Competence management and expert identification are activities within the field of knowledge management which aim to find out *who* knows *what* [1]. This information can serve various purposes, e.g., to find the right employees to staff new projects [2], to match positions with skills [3], or to support software maintenance [4].

In distributed environments or in larger development teams knowledge about *who* wrote a particular piece of code, *who* knows about a particular set of classes, *who* is responsible for a particular requirements document is essential. According to [5], software developers apply just as much effort and attention determining whom to contact in an organization as they do getting the job done.

To partly solve this problem, this paper proposes a measurement framework to extract knowledge about *who* knows *what* about software development artifacts (such as source code, documents, slides, spreadsheets, etc.) non-invasively, i.e., without the need by the developers to spend time on documenting their knowledge within a knowledge management system.

The information that is extracted builds on the idea that the programmer's activity, i.e. the adding, modifying, deleting, reading of code is an indicator of the knowledge that the programmer has about that part of the code [6].

Other approaches such as expertise recommenders, which suggest who has expertise in particular parts of the program also base their recommendations on this assumption. Tools such as Expertise Recommender [7], EEL [8], and Expertise Browser [9] make recommendations based on commits to source code repositories.

The objective of the expert identification measurement framework is twofold: the primary goal is to track the time spent editing the artifacts that are created during the software development process, such as documents, source code, spreadsheets, etc. to infer the user's knowledge on that artifact. Additionally, data that describes the artifact being accessed is also collected to allow the retrieval of an artifact using its properties as search criteria.

2. Related work

Examples of existing implementations can be categorized within two groups depending if the knowledge about *who* knows *what* has to be provided by the users of the system or if it is extracted from other sources.

Examples of tools of the first group are the StepStone Skills & Competency Management Module [10] or SAP ERP Human Capital Management [11] in which employees maintain their own or the skill profiles of their subordinates.

A common problem of knowledge engineering is how to generate knowledge with as less effort and resources as possible [12]. Asking employees to maintain their knowledge profile takes time which means that it will generate costs since if experts spend time sharing knowledge, they will be less productive [13].

To overcome this barrier to adopt knowledge management, tools that try to extract knowledge automatically from existing artifacts *who* knows *what* were developed. Two examples are AnswerGarden [14] which creates a knowledge repository storing the questions and answers exchanged between help desks and their clients and ActiveNet [15] which extracts knowledge

from the e-mail traffic, instant messaging, and digital workspaces that employees use in their every days' work.

Proposals like the "Knowledge Dust to Pearls Approach" [13] build on the AnswerGarden approach to refine the collected knowledge into "experience pearls" that are collected in form of an "experience base" for the purpose to be reused for the planning of future activities.

3. Measurement framework

To extract the desired knowledge from the ongoing software development process, we developed a measurement framework that is able to identify *on which artifact a user is currently working on*, to read *properties of the artifact currently accessed*, *how long* it is accessed, and to store this information in a central database. It was our focus during development that all steps can be done non-invasively, i.e. without the need for the developer to interact with the knowledge management system.

We use the period of time spent on a part of the system as an indicator of the knowledge of a developer. As a developer works on a system, he or she gains knowledge about the domain of the system, the development process used to build the system, and the design and implementation of the system [16]. Previous studies such as [6], or implementations such as [7] and [9] have shown that activity as a knowledge indicator can be used to build a model of what a programmer knows about a code base.

Currently our measurement framework can identify artifacts accessed using the Microsoft Office suite [1] (Word, Excel, PowerPoint, Visio, Frontpage, and Outlook), the OpenOffice.org office suite[2] (Writer, Calc, and Impress), and software development environments such as Microsoft Visual Studio[3], Eclipse[4], NetBeans[5], and IntelliJIDEA[6].

The mentioned applications allow to read the currently accessed artifact through a provided API. We developed a set of measurement probes (one per application) that constantly poll these applications about the current artifact accessed. As soon as the reported value changes, the amount of time passed since the last change is written to the local log file together with the current date, time, user name and name of the artifact accessed. In regular intervals the so collected log is transferred to a central server.

We assume that the API of the application to observe provides a function *getCurrentArtifact()* that we use to access the current artifact modified by the user (in most of the cases this is the artifact that is currently on focus). The following pseudo code describes how one measurement probe (i.e. an application of our measurement framework connected to the API of an application to observe) collects data.

```
initialize user with the current user
initialize app with the application name
initialize artifact using getCurrentArtifact()
initialize start with the current date and time

while application to observe is running
   // we want to collect data with the
   // granularity of one second so that
   // our application does not consume
   // too many resources on the machine
   // of the developer
  wait for 1 second
  if getCurrentArtifact() <> artifact then
    set now to the current system date and time
    set duration to now - start

    append user, app, artifact,
      start, duration  to the local log file

    set artifact = getCurrentArtifact()
    set start = now
  end if
end while
```

The pseudo code above shows how the probe constantly polls the observed application and generates an entry in the local log file if the current artifact changes. The so obtained activity log files are transferred to a central database where all data is stored. The three steps, artifact identification, local data caching, and data transfer are illustrated in figure 1.

What we consider as the concrete artifact depends on the application: within Microsoft Office and OpenOffice it is a file. This means that we track the accesses and modifications of the properties of files. Within programming environments we consider the method as the artifact, i.e., the time spent editing, adding, deleting a method is the highest granularity of the data that is collected. This data can be aggregated later e.g., at the class, namespace, or file level.

The measurement unit of time is *seconds*, which is then aggregated to hours or days on reports.

The result of the data collection step consists of artifacts, properties of artifacts, and the sequence of accesses on the artifacts (see figure 2). In this way, the described measurement framework collects *when, who* accesses *which artifact,* and – to allow the filtering of artifacts – logs properties that describe the artifacts accessed by the user.

Collecting data about the ongoing software production without the direct intervention of the developer has been proposed also in other contexts than Knowledge Management: e.g., the HackyStat tool [17], which can be used to collects metrics about produced artifacts to better control the software development process and to give

[1] Microsoft Office. http://office.microsoft.com
[2] OpenOffice.org. http://www.openoffice.org
[3] Microsoft Visual Studio, http://msdn2.microsoft.com/vs2008/products
[4] Eclipse.org, http://www.eclipse.org
[5] Sun NetBeans, http://www.netbeans.org
[6] JetBrains IntelliJIDEA, http://www.jetbrains.com/idea

feedback to developers about the impact of their work on quality properties of the developed software.

4. Data analysis

To help users searching for experts within the system described above, we categorize the collected time spent editing artifacts according to different criteria.

For example, if the source code of a specific company is organized in such a way that the namespace indicates the component of the software system, it is reasonable to group the time spent per artifact by namespace. In this case, knowing the namespace, i.e. the component, helps to find who dedicated the most time accessing artifacts within that namespace, i.e. the employee with the most knowledge of this component, the expert [6].

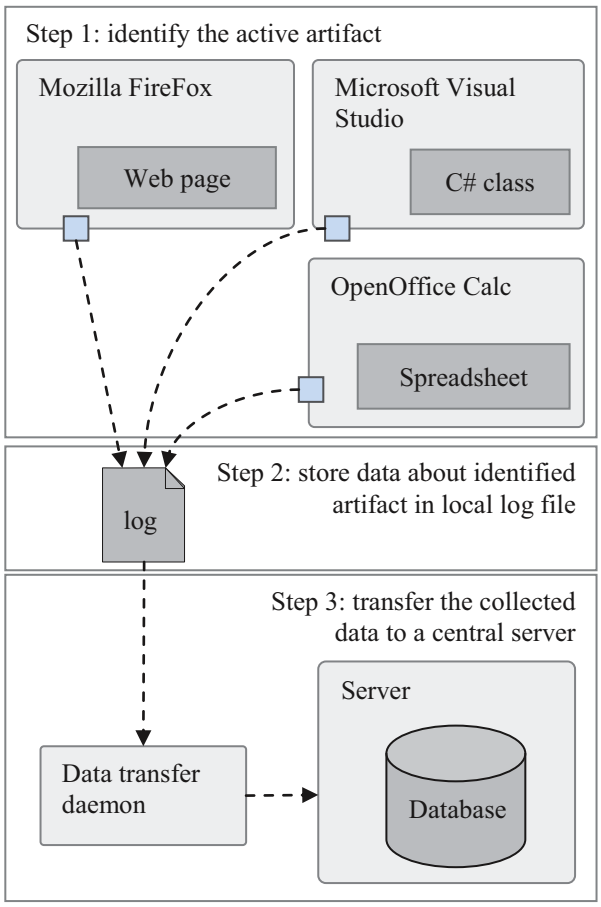


Figure 1. Overview of the framework

Other examples for classifying the access duration are the project name, prefixes of class names (e.g., "test" to find the testing experts), or types of documents.

To be able to change the rules easily, we use Prolog rules to define how the data should be classified (using Interprolog [18] as a bridge between Java and Prolog).

Within Prolog we define a set of predicates that correspond to the properties collected by the measurement framework which allow access to the table "properties" (see figure 2) within Prolog.

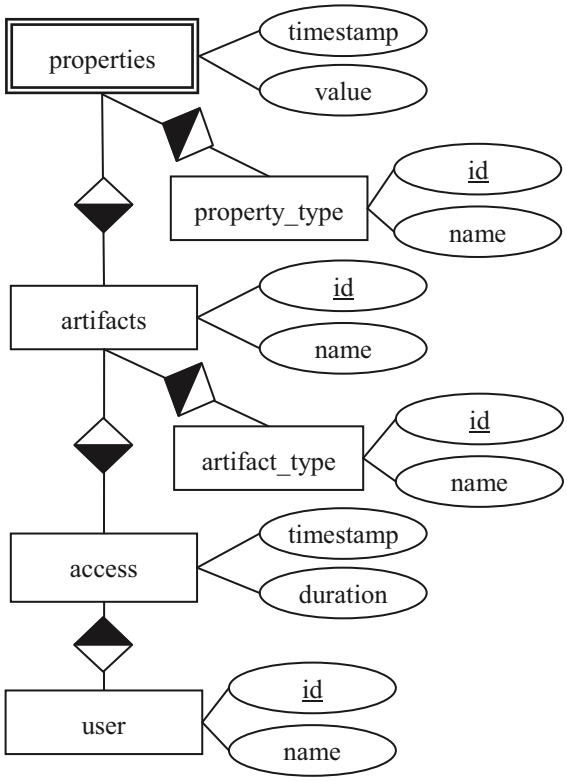


Figure 2. EER diagram of the measurement framework

Currently the following predicates are available for the use within Prolog rules:

a) *path(X, N)*: true, if the artifact with the id *X* is a file stored within the folder with the name *N* (*N* is specified as a Prolog array, i.e. ['c', 'a', 'b'] for the namespace "c:\a\b");
b) *file(X, N)*: true, if the artifact with the id *X* is a file with the name *N* (without path);
c) *class(X, N)*: true, if the artifact with the id *X* represents a class with the name *N*;
d) *method(X, N)*: true, if the artifact with the id *X* represents a method with the name *N*;
e) *namespace(X, N)*: true, if the artifact with the id *X* is contained within the namespace *N* (*N* is specified as a Prolog array, i.e. ['a', 'b'] for the namespace "a.b").
f) *access(Y, U, X, S)*: true, if the access with the id Y by the user *U* to the artifact *X* lasted *S* seconds.
g) *access_date(Y, D):* true, if the access with the id *Y* occurred on the date *D*.

To define a classification of the artifacts, we expect that the predicate *classification_artifact(T, X, C)* exists and that it returns true if an artifact with the id X is

contained in the class with the name C according to the classification criteria with the name T.

To classify the records stored within the table *access* (e.g. to classify certain accesses within a certain time range to a specific category) we expect that the predicate *classification_access(T, A, C)* exists and that it returns true if an access with the id A is contained in the class with the name C according to the classification criteria with the name T.

The resulting classifications are cached within the database so that they can be easily queried using SQL. Figure 3 shows the EER diagram of how the classifications of artifacts and accesses to these artifacts are stored within the database: classifications can be made according to different criteria. For example, it is possible to classify artifacts according to their importance and according to their size.

All criteria *T* that the classifying Prolog predicates *classification_artifact* and *classification_access* return are stored within the table *classification_criteria*. All classes *C* used within the two classification predicates are stored in the table *classification_classes*.

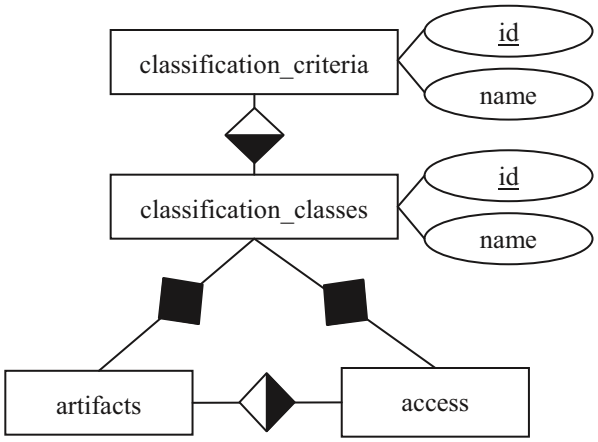


Figure 3. EER diagram of how the classifications of artifacts and accesses are stored in the database

5. Examples

In the following two examples we will show how the described measurement framework can be used to address expert identification and skill management issues.

5.1. Example 1

In the following example we describe the analysis of experts within a case study carried out for a company in the domain of industrial automation, which, for confidentiality, in the following we call "Acme".

Acme's IT department consists of about 50 employees, 23 of them are developing software for internal use.

We installed our measurement framework and collected the time spent per method, class, namespace, and file. Within Acme, the developers agreed that the development effort should be grouped on the namespace level, considering only the first two packages since the first package for them represents always the name of the project and the second package the name of a main component.

If, e.g. a class is contained within the namespace *a.b.c.d*, we attribute all the time spent editing in this class to the class *a.b*.

Therefore, as rules, we defined the predicate *first_two(X, H1, H2, R)* so that it is true if *H1* and *H2* are the first two elements of the array *X* and *R* is the remainder of the array:

```
first_two(X, H1, H2, R):-namespace(X, Y),
  [H1|T1] = Y, [H2|R] = T1.
```

Now, *classification_artifact(T, X, C)* can be defined as follows:

```
classification_artifact(T, X, C) :- T=expert,
  first_two(X, H1, H2, R),
  concat(H1, '.', P1), concat(P1, H2, C).
```

This means that for the type of classification *expert*, we consider an artifact with the id X part of the class C, if the first two packages of the namespace of the artifact X correspond to the name of C.

To visualize the data obtained using our measurement framework, every tool able to connect to a database using JDBC can be used, we used OpenOffice Calc[7] to query the database and to generate a pivot table of the format as shown in figure 4.

In this table, all classification items accessed within the analyzed period are shown as lines, the users accessing these classification items as columns. Within the pivot table the single values represent the sum of time spent by the specific user on a specific classification item.

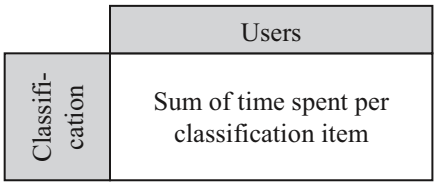


Figure 4. Schema of the pivot table to visualize the effort distribution

In this way, calculating the ith largest value of time spent on each classification item, we obtain the ith expert of that classification item. Within Acme, we used percentages (the time spent in relation to the time spent by the top expert) to ease the understanding of the resulting pivot table.

[7] OpenOffice.org Calc, http://www.openoffice.org/product/calc.html

In the example shown in table 1, user 2 has almost the same amount of experience as user 1 considering classes within namespaces starting with *project1.a*, but he is the only one that has experience with classes within *project2.c*.

Within our case study we formatted the table above to ease the understanding of the data for the user: we color the cells in dark green if the value is above 90% and in light green if it is above 50%.

The column "Experts" shows the number of users within the current line with values above 50%. If in this column the number of users is equal to 1, this means that there is only one user that knows about that part of the code. Within the case study we colored these cells in red.

		Users			
		User1	User2	User3	Experts
Classification	project1.a	100%	90%		2
	project1.b	10%		100%	1
	project2.c		100%		1
	project3.d	10%	100%		1

Table 1. Example of pivot table representing the knowledge of each user about the code within a classification item

To summarize, using the mentioned measurement framework and a tool to query the obtained data such as OpenOffice Calc, it is possible to:

a) determine the ith expert of a specific part of the code, assuming that the time spent in adding, modifying, and updating reflects the knowledge of the code,
b) to show the experience of a specific user,
c) to evaluate the knowledge gap that will occur if a specific user will leave.

Point c) addresses the problem that when a person with critical knowledge leaves an organization, this creates severe knowledge gaps. It is crucial to understand *what knowledge* is lost to prevent valuable knowledge from disappearing. Additionally, knowing which knowledge disappeared can help to decide which skills are needed for the new employee and on which areas he or she should work on.

5.2. Example 2

Acme is using Extreme Programming [19] as software development methodology.

Within agile methodologies, the focus lies on "working software over comprehensive documentation" [20]. The produced source code is considered the most valuable asset, representing the knowledge of the development team. This knowledge has to be shared among team members using practices like "pair programming" (meaning that all code has to be programmed in two) or "common code ownership" (meaning that the entire team is responsible for the source code and that everybody has the right to modify everything) [19].

		Users		
		User1	User2	User3
Classification	project1.a	100%	90%	20%
	project1.b	80%	80%	100%
	project2.c	90%	100%	80%
	project3.d	70%	100%	80%

Table 2. Example of pivot table representing the knowledge of each user about the code within a classification item (considering only the last 6 months)

In such contexts it is important that *all* know *everything* about the produced code. A measurement framework as described here can be used to determine the knowledge of different developers about different subparts of the code and could be extended to recommend *who* should do pair programming with *whom* to optimally distribute the knowledge.

In the case of Acme, we decided to consider only the time spent in the last half year as an indicator of experience. If e.g., a programmer for more than half year did not dedicate time in the development of a specific component he or she had to do pair programming with one of the developers currently working on that part to be up to date with the last modifications.

For this we defined the predicate *classification_access(T, A, C)* as follows:

```
classification_access(T, A, C) :- T=expert,
  access_date(A, B), parse_time(B, C),
  get_time(D), E is D-C, E < 259200, C=pp.
```

This means that for the type of classification *expert*, we consider an access to an artifact with the id *X* part of the class *pp* (pair programming), if the access date (obtained as a string *B* and converted to the Prolog timestamp *C*) is not less than six months (259200 seconds) ago.

Within our SQL query we sum up all the accesses belonging to the class *pp*.

As in the example 1 we used OpenOffice Calc to visualize this data. The resulting table is formatted in the same way (see table 2 for an example), just that now the data shows *who worked on which classification item (i.e. the first to elements of the namespace) during the last 6 months.*

In the example in table 2 it is visible that user 3 should to pair programming either with user 1 or user 2 for the next requirement on the component *project1.a*.

6. Acknowledgements

I thank Tadas Remencius for his support on this article.

7. References

[1] Rus, Ioana and Lindvall, Mikael., "Guest Editors' Introduction: Knowledge Management in Software Engineering." IEEE Software, 2002, Issue 3, Vol. 19, pp. 26-38.

[2] Becerra-Fernandez, Irma., "Searching for experts on the Web: A review of contemporary expertise locator systems." ACM Trans. Inter. Tech., New York, NY, USA : ACM, 2006, Issue 4, Vol. 6, pp. 333-355.

[3] Becerra-Fernandez, Irma., "Facilitating the Online Search of Experts at NASA using Expert Seeker People-Finder." [ed.] Ulrich Reimer. s.l. : CEUR-WS.org, 2000. PAKM. Vol. 34.

[4] Sarkar, Santonu, Sindhgatta, Renuka and Pooloth, Krishnakumar., "A collaborative platform for application knowledge management in software maintenance projects." New York, NY, USA : ACM, 2008. Compute '08: Proceedings of the 1st Bangalore annual Compute conference. pp. 1-7.

[5] Perry, Dewayne E, Staudenmayer, Nancy and Votta, Lawrence G., "People, Organizations, and Process Improvement." IEEE Softw., Los Alamitos, CA, USA : IEEE Computer Society Press, 1994, Issue 4, Vol. 11, pp. 36-45.

[6] Fritz, Thomas, Murphy, Gail C and Hill, Emily., "Does a programmer's activity indicate knowledge of code?" New York, NY, USA : ACM, 2007. ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. pp. 341-350.

[7] McDonald, David W and Ackerman, Mark S., "Expertise recommender: a flexible recommendation system and architecture." New York, NY, USA : ACM, 2000. CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work. pp. 231-240.

[8] Minto, Shawn and Murphy, Gail C., "Recommending Emergent Teams." Washington, DC, USA : IEEE Computer Society, 2007. ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops. p. 5.

[9] Mockus, Audris and Herbsleb, James D., "Expertise browser: a quantitative approach to identifying expertise." New York, NY, USA : ACM, 2002. ICSE '02: Proceedings of the 24th International Conference on Software Engineering. pp. 503-512.

[10] , Skills & Competency Management Software by StepStone Solutions. [Online] StepStone . [Cited: March 11, 2008.] http://www.stepstonesolutions.com/Solutions/Skills_Competency_Management/Skills_Competency_Management.php.

[11] SAP ERP Human Capital Management. [Online] SAP. [Cited: March 11, 2008.] http://www.sap.com/solutions/business-suite/erp/hcm/index.epx.

[12] Rus, Ioana, Lindvall, Mikael and Sinha, Sachin Suman., *Knowledge Management in Software Engineering.* DACS State-of-the-Art-Report, The Data & Analysis Center for Software (DACS) is a Department of Defense (DoD) Information Analysis Center (IAC). 2001. http://www.cebase.org:444/umd/dacs_reports/kmse_-_nicholls_final_edit_11-16-01.pdf.

[13] Basili, V, et al., "An Experience Management System for a Software Engineering Research Organization." Washington, DC, USA : IEEE Computer Society, 2001. SEW '01: Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop. p. 29.

[14] Ackerman, M S and Malone, T W., "Answer Garden: a tool for growing organizational memory." New York, NY, USA : ACM, 1990. Proceedings of the ACM SIGOIS and IEEE CS TC-OA conference on Office information systems. pp. 31-39.

[15] , ActiveNet - Facilitating Real-Time Collaboration. [Online] Tacit Software. http://www.tacit.com/products/activenet/technology.html.

[16] Susan Elliott Sim, Richard C. Holt., "The Ramp-Up Problem in Software Projects: A Case Study of How Software Immigrants Naturalize." 1998. 20th International Conference on Software Engineering.

[17] Johnson, Philip M, et al., "Beyond the Personal Software Process: metrics collection and analysis for the differently disciplined." Washington, DC, USA : IEEE Computer Society, 2003. ICSE '03: Proceedings of the 25th International Conference on Software Engineering. pp. 641-646.

[18] Calejo, Miguel., "InterProlog: Towards a Declarative Embedding of Logic Programming in Java." [ed.] José Júlio Alferes and João Alexandre Leite. Lisbon, Portugal : Springer, 2004. Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lecture Notes in Computer Science 3229. Vol. 3229, pp. 714-717. DOI 10.1007/b100483. ISBN 3-540-23242-7.

[19] Beck, Kent and Cynthia, Andres., *Extreme Programming Explained. Embrace Change.* 2nd Edition. Amsterdam : Addison-Wesley Longman, 2004. ISBN 0321278658.

[20] Beck, Kent, et al., Manifesto for Agile Software Development. [Online] 2001. http://agilemanifesto.org.

Using XML Patterns to Guide Perturbation Based Testing of Web Services

Paulo N. Cruz Filho
paulonei@inf.ufpr.br
DInf-UFPR - Federal University of Parana
Brazil CP: 19081, CEP: 81531-970

Silvia Regina Vergilio
silvia@inf.ufpr.br
DInf-UFPR - Federal University of Parana
Brazil CP: 19081, CEP: 81531-970

Abstract

Web Services contribute to decrease costs and solve integration problems in web applications. On the other hand, they bring some complications to test activity and demand specific testing techniques and tools. In this context, the perturbation based approach has been successfully explored. Perturbation operators modify XML and SOAP messages that are then used as test data. However, the set of proposed operators are not complete. Considering this fact, in this paper, a new perturbation approach is introduced based on XML patterns associated to UML models. In this way, the approach considers semantic aspects related to other kind of faults. The implementation aspects and evaluation results of the introduced approach are also discussed.

Keywords: XML, fault-based testing, perturbation testing

1 Introduction

Web Services (WS) are software programs that operate independently to offer services over the Internet to other software programs, including web applications and other WS [16]. They are developed to allow interoperability among technologies, as well as, protocols, platforms and operating systems. WS are a well-known implementation of the Service Oriented Architecture (SOA), which conceptually defines a structured data exchanging model, providing the applications the ability to be loosely coupled with limited knowledge of each other implementations.

With the advent of Internet, the use of WS is crescent. They contribute to minimize costs and to solve integration problems in web applications. However, they demand specific testing techniques and tools and have been considered a challenge by many authors [7]. This fact is due to their specific characteristics. They are more widely distributed and heterogeneous. They involve multiple standards and protocols than traditional software, moreover, the absence of a user interface makes it harder to apply a test procedure, due to the loss of controllability. Moreover, WS admit changes of data flow among software components at execution time, what is called dynamic integration.

Bloomberg [2] presents several aspects involving Web Service testing. For example, WS make extensively use of XML technology [14] to exchange data among applications, thus, testing interactions based on XML messages is a very important aspect to be tested, contributing to Web Service quality assurance. The SOAP protocol [15] (basis of the Web Service technology) is an XML message that carries information and is responsible for invoking WS over the Internet.

Considering these different aspects, some works address Web Service testing [1, 3, 9, 10, 11, 13]. In [11] a framework that converts WSDL specifications into test scenarios is described. In [3] Finite State Machines are used to model and test the WS behavior. Heckel and Lohmann explore the use of contracts [9]. In [1, 10, 13] a method based on data perturbation of XML and SOAP messages is explored. XML messages are modified and used as test data for testing the interaction between pairs of WS. The method can be applied without knowledge about the implementa-

tion of the WS and has presented promising results in terms of revealed faults. However, the explored perturbation operators consider only syntactic aspects to generate the test data. Semantic aspects related to the data specification need to be considered because they can generate test data related to other kind of faults. With the objective of improving efficacy in terms of revealed faults, this paper introduces an XML pattern driven data perturbation approach.

The introduced approach makes use of UML-XML mapping techniques [5] and XML structural patterns [8]. To perturb an original XML message, the XML vocabulary, given by a schema and its correspondent UML model, is analysed. According to the pattern found in the schema, the original message is perturbed and used as test data. The idea is to consider the meaning of XML vocabularies to generate test data and to reveal specific faults related to the semantics of the XML messages. In this sense, the approach is adequate for applications that use XML vocabularies modeled with UML language and can be viewed as complementary to the existent perturbation testing approaches.

The remaining of this paper is organized as follows. Section 2 shows related works on WS testing. Section 3 overviews XML-UML mapping techniques and patterns used in the introduced perturbation approach. Section 4 describes and illustrates the XML pattern guided approach. Section 5 experimental results from an evaluation study and comparisons with traditional operators. Section 6 concludes the paper.

2 Web Service Testing

There are many issues involving Web Service testing [2], for example: testing SOAP messages; testing WSDL files; testing the publish, find and bind capabilities of a SOA; testing Web Service consumer and producer emulation; testing synchronous and asynchronous capabilities; etc. Some works address these issues. Tsai et al [11] describes a framework that converts WSDL specifications into test scenarios. In [12], a method for test data generation based on the topology of WS is proposed. Heckel and Lohmann [9] explore the use of contracts. Bultan et al [3] propose the use of Finite State Machine to model and test the behavior of composite WS. The model is composed by multiple peers that communicate with asynchronous messages. However, the test can also be conducted considering a peer-to-peer model. For this kind of model, the works [1, 10, 13] address the use of modified XML and SOAP messages to test WS.

In [10], Offutt and Xu have introduced an approach based on data perturbation for Web Service testing. The idea is to modify request messages by using some mutation operators. It is similar to mutation testing [6], however, the difference is that the mutants are actually the test cases to be used to test the Web Service. Similar works that have the same objective are [1, 13]. In [1] a tool, named SMAT-WS, is described and used in the test of nine WS.

Perturbation Based Testing of WS has been successfully applied. Experimental results show that this approach is fault-revealing. It can be considered a promising testing approach to test interactions of XML based components of SOA applications such as WS. It is independent on how the application (or service) is implemented. Another advantage is that test data generation can be easily automated. A disadvantage is that it is not possible to determine whether the set of operators is complete. The proposed operators, for example, only consider syntactic aspects of XML documents. Semantic aspects related to the data specification need to be considered. In the Section 4 we introduce a new kind of perturbation approach, guided by XML patterns.

3 XML Patterns Based on UML

The use of UML to define XML vocabularies presents some advantages: 1) UML allows the visualization and representation of XML structures through standardized diagrams; 2) ability to capture the semantics of the XML model; and 3) easy reading for humans. Some works address XML-UML mapping techniques. In our work we use the approach proposed by Carlson [5] and supported by the tool hyperModel [4]. The main concepts from the approach are illustrated in Figure 1. The UML classes are mapped to *complexType* definitions. In addition to, an element is declared, with the same type of the *complexType* of the class. The attributes and associations of the classes are mapped to XML elements. The used content model was <sequence>, which makes possible to list the class elements, with their multiplicity restrictions. The UML inheritance is mapped using XML Schema inheritance.

In [8], the idea of using UML models to establish XML patterns is introduced. The idea is to provide patterns to XML documents that are expressed with visual models. Two kinds of models are explored: design patterns related to Web applications and patterns more directly related to the structure of UML models used to model XML vocabularies.

Table 1 displays a summary of the main XML patterns. To illustrate the patterns, we will use the simplified XML vocabulary from OASIS Universal Busi-

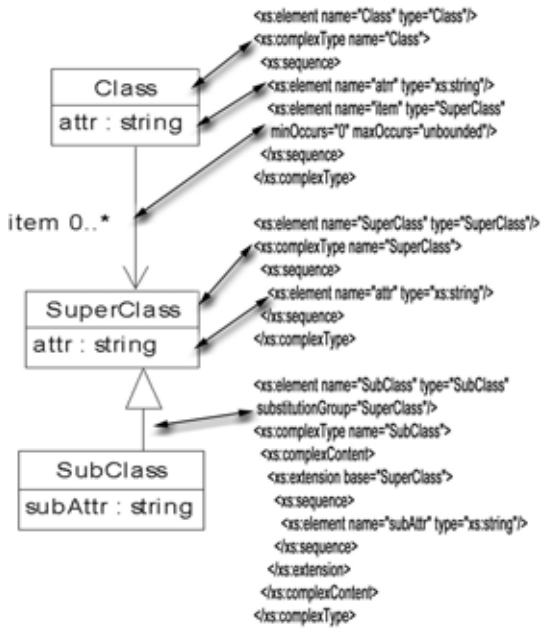


Figure 1. UML-XML mapping proposed by Carlson

Table 1. XML Structural Patterns

Pattern	Description
homologous association	expresses the association between two concepts in different contexts
recursive association	models a relationship among elements defined in a recursive way
multiple reference	models, as a singular class, an element that can be referenced by multiple classes
homologous derived	establishes a group of derived classes from a same basis class, which can present class relationships amongst themselves

ness Language (UBL) [4] shown in Figure 2. It models the transport of hazardous items, in which the temperature control is fundamental. In this fragment, we identify two groups of Homologous Associations. The first includes the associations *MaximumTemperature* and *MinimumTemperature*, the second one contains *EmergencyTemperature* and *FlashpointTemperature*. This vocabulary also presents the pattern Multiple Reference. Different classes refer to class *Temperature*.

4 Using XML Patterns for Test Data Generation

In this section, we explore the use of XML structural patterns for test data generation and introduce perturbation testing based on patterns. The idea is to consider the meaning of XML vocabularies to generate test data and to reveal specific faults related to the semantics of the XML messages. The approach can be applied in applications that use XML vocabularies modeled with UML.

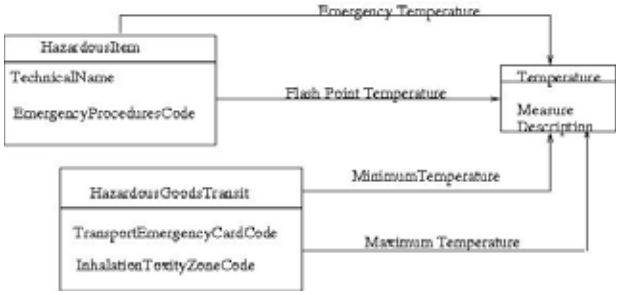


Figure 2. OASIS Universal Business Language Vocabulary Fragment

Initially, we introduced and implemented a set of four perturbation operators. The operators have the same name that the original pattern. To apply an operator, the correspondent pattern is identified in the XML vocabulary associated to the WS being tested, given by a schema and its respective UML model. In this way, we identify an element. According to a pattern, we decide the transformation to be applied and the element will be changed, removed, etc. Values in the message are changed by other possible ones present in a library of vocabulary instances. The perturbed message is then used as test data. The operators are:

a) HA (Homologous Associations): changes attribute values of a class related to an association A by the attribute values of the same class related to the homologous association B.

Consider again the vocabulary from Figure 2. The pattern Homologous Associations shows two different classes linked through two or more associations, which reveal the specific meaning of the relationships. The perturbation in the XML message, defined considering the pattern HA, consists of changing the data that corresponds to an association by the data related to other arbitrary homologous association. This way, the semantics of the message is altered to a different semantics, but possibly very close and coherent regarding the original meaning. For example, the application of HA can generate a test message that involves the *MaximumTemperature* and *MinimumTemperature* associations. A message relative to the hazardous items transport could present the following fragment to define the temperatures:

```
<HazardousItem>
```

```
  <TechnicalName> Plumbum Nitrate II
  </ TechnicalName>
        ...
  <HazardousGoodsTransit>
    <TransportEmergencyCardCode> 51
    </ TransportEmergencyCardCode>
        ...
    <MaximumTemperature>
      <Measure> 27 </ Measure>
        ...
    </ MaximumTemperature>
    <MinimumTemperature >
      <Measure > 10 </ Measure>
        ...
    </ MinimumTemperature>
  </ HazardousGoodsTransit >
</ HazardousItem >
```

Suppose that the chosen element to be perturbed is *MaximumTemperature* that has the measure value 27 for this instance. The developed perturbation algorithm will substitute this value by the content of a *MinimumTemperature* element, found in some instance of a given XML document. For example, consider that there is an instance with value -7 for *MinimumTemperature*, such as:

```
    ...
    <MinimumTemperature >
       Measure > -7 </ Measure>
        ...
    </ MinimumTemperature>
    ...
```

When perturbing the previous instance, the value of *MaximumTemperature* of the item being perturbed will be substituted by the value of *MinimumTemperature* of the second instance. The resulting instance is:

```
...
    <MaximumTemperature>
      <Measure> -7 </ Measure>
        ...
    </ MaximumTemperature>
    <MinimumTemperature >
       Measure > 10 </ Measure>
        ...
    </ MinimumTemperature>
       ...
```

In the perturbed instance the value for the maximum temperature is smaller than the value of the minimum temperature. This message can produce a failure, if such situation was not considered in the implementation of the web service being tested.

b) MR (Multiple References): changes attribute values of a class A related to a reference of class B by attribute values of A related to a reference of another class C.

Figure 3 presents a fragment extracted from UBL [4]. The class *Country* is referred by three other

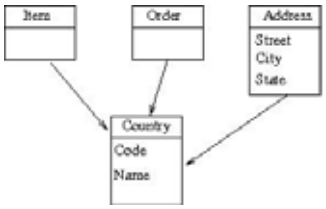


Figure 3. Multiple Reference Perturbation

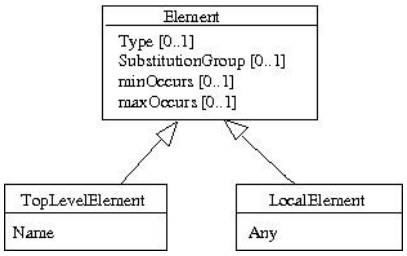


Figure 4. Homologous Derived Perturbation

classes. An example of perturbation is to change the value of attribute *Name* from class *Country* correspondent to the address of a client by other value correspondent to the source country of an item.

c) HD (Homologous Derived): changes attributes values of a derived class by common attribute values of the homologous derived class.

Consider the fragment of W3C XML Schema for Schemas vocabulary [4] of Figure 4. There are two derived classes for *Element*. The common attribute value of *minOccurs* (or *maxOccurs*) of *TopLevelElement* is changed by the attribute value of *LocalElement*.

d) RA (Recursive Association): in general, a document that follows this pattern corresponds to a tree hierarchy. The operator changes this hierarchy. Each element in the tree is changed by its predecessor (or successor) in the hierarchy. Consider the fragment of eBay XML API vocabulary [4] of Figure 5 and the following message:

```
<SOAP-ENV:Envelope ...>
 <SOAP-ENV:Body>
  <ns1:processCategory
      xmlns:ns1=''http://soapinterop.org/''>
   <name xsi:type=''xsc:string''>processors</name>
        ......
  </ns1:processCategory>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope ...>
```

The store is divided in categories and each derived category has its own subdivisions represented in a tree. A computer store has a division of components, which is divided into monitors and processors categories.

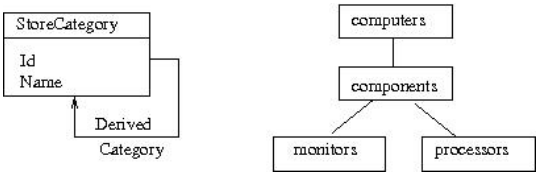


Figure 5. Recursive Association Pattern

An example of perturbation for the above message that uses RA operator is to change the value "processors" either by "monitors" or "computers", which are other categories in the hierarchy.

5 Case study

To make possible a better analysis of the proposed data perturbation guided by XML patterns, and to allow its practical application, a tool -called PDDP (Pattern Driven Data Perturbation Tool) - was developed. The fundamental artifacts for PDDP are: the SOAP message in which the perturbations are applied; the vocabulary definition used in the components communication; a library of vocabulary instances, used to derive the perturbed messages.

The initial SOAP message is used as a model to create the perturbed messages, where we have at least one modification of an element in the XML sub-tree that is encapsulated in the element *Body* of the SOAP protocol. The vocabulary definition is supported by tool hyperModel, driven to the Carlson mapping method. The XML Schema generated by hyperModel is used as an input for PDDP. The instances in the library that obeys the defined vocabulary are the source of elements used for the generation of perturbed messages.

A study was accomplished with PDDP to evaluate the applicability of the proposed perturbation approach and the efficacy of the operators in comparison with the traditional ones. To do this, we used the same system of WS used in the experiment reported in [1], as well as its results.

The system was tested during its development and the found faults are not seeded. It is composed by nine WS and integrates other systems that involve financial, administrative and structural government controls. Each Web Service was submitted individually to PDDD. During the execution of the tests we had free access to the database, so that results of the operations could be verified. Two input XML files were supplied: an XML Schema with the vocabulary definition, generated by hyperModel tool; and an XML file with instances of valid elements, according to the XML Schema. For each Web Service a valid initial message was given, from which the perturbed messages were derived.

Table 2. Test Cases and Faults

Web Service	Test Cases	Effective Test Cases	Found Faults
WS1	16	4	2
WS2	19	2	1
WS3	9	1	1
WS4	13	3	1
WS5	9	0	0
WS6	9	1	1
WS7	14	2	2
WS8	11	2	2
WS9	12	3	2
Total	112	18	12

Table 3. Operator Efficiency

Traditional Operators	ML	I	N	BE	IN
	14.81	29.63	38.89	6.61	4.65
	VI	S	B	U	
	25.56	35.19	16.67	29.63	
Semantic Operators	HA	RA	HD	MR	
	29.16%	27.77%	0%	20.14%	

Table 2 presents, for each Web Service, the number of test cases generated, the number of test cases that revealed faults and the total of faults found. From this result, we can analyse the contribution of each operator by analysing the total number of generated test cases and the number of revealed faults. MR operator presented the greatest contribution (72.2%). It is the most common pattern found in the model; the operators HA (8.3%) and RA (11.1%). On the other hand, the operator HD did not contribute to reveal faults. However this low contribution can be explained by the nature of the used WS. The parameters perturbed by the HD pattern correspond to a user login, which is related to a password. It is observed that the user validation mechanism is simple and did not present faults.

Table 3 presents results comparing the semantic and traditional operators.[1] The efficiency is defined as the quotient between the number of test cases that revealed faults for a specific operator and the total number of test cases generated by the same operator.

The semantic operators present similar efficiencies. This does not happen with the traditional ones. If we consider the general efficiency (the great average), the efficiency of the semantic operators (21.2%) is comparable to the efficiency of the traditional ones (19.2%).

Other point to be considered is the type of faults found by each group of operators. The operators have distinct characteristics and because of this, reveal dif-

[1]This table shows the following traditional operators (results extracted from [1]): Mod_ Len (ML), Incomplete (I), Null (N), Boundary Extension (BE), Inversion (IN), ValueInversion (VI), Space (S), Boundary (B) and Unauthorized (U).

ferent kind of faults. The traditional operators revealed faults related to validation of parameters. The semantic ones, generates valid messages, which are test cases with greater probability of revealing faults related to the application business logic. This can explain the reasons why the number of faults found by the semantic operators was smaller than that revealed by the traditional operators. To reveal faults related to the application businesses logic is something more ambitious and more difficult.

6 Conclusions

This paper explored the use of XML structural patterns in the test activity. Data perturbation techniques were extended and applied to test the communication of components that exchange XML messages. The result was the proposition of the pattern driven data perturbation approach.

The traditional perturbation operators only consider syntactic aspects: limit values, restrictions about data types and multiplicity definitions. The use of patterns to guide perturbation testing allows the generation of test data considering the meaning of the XML vocabulary. So the chance of generating significant test data is greater, as well as, the probability of revealing other kinds of faults.

We implemented a supporting tool that makes the use of the approach practical and shows its applicability. An evaluation experiment was conducted with a real system. The efficiencies of the introduced operators are very similar. However, the number of generated test cases and revealed faults are related to the characteristics of the vocabulary and found patterns.

When compared with traditional operators, the results point out that both groups of operators are complementary. The semantic operators contributes to reveal other kind of faults. They reveal faults related to the application businesses logic validation specially to the communication vocabulary. The traditional ones reveal faults more related to the parameter validation.

The pattern based perturbation testing is adequate for applications that use XML vocabularies modeled with UML language. In this work, the approach was explored for WS testing. However, it can be applied in other applications that not necessarily exchange message over the Internet or network. For example, to test the communication among modules of a system through XML messages defined according to some vocabulary. A possible extension for this approach is to explore the use of other models, such as ER.

References

[1] Almeida Jr, L.; Vergilio, S.R. Exploring Perturbation Testing for Web Services. In: IEEE Intern. Confer. on Web Services, 717-726, October 2006.

[2] Bloomberg, J. Report: Testing Web Services. White paper of report for Parasoft SOAPTest, 2002 (accessed January 2006). http://www.parasoft.com/jsp/templates/misc/soap/web_services_excerpt.pdf.

[3] Bultan, T.; Fu, X.; Hull, R.; Su, J. Conversation specification: A New Approach to Design and Analysis of e-service Composition. In: 13th Intern. World Wide Web Confer. 2003.

[4] Carlson, D. hyperModel Tool. http://www.xmlmodeling.com/.

[5] Carlson, D. Modeling XML Applications with UML. Addison-Wesley, 2001.

[6] DeMillo, R. A.; Lipton, R.J.; Sayward, F.G. Hints on Test Data Selection: Help for Practicing Programmer. IEEE Software, 11:34-41, April 1978.

[7] Di Lucca, G. Testing Web-Based Applications: the State of the Art and Future Trends. In: QATWBA'05 - Workshop of the Intern. Computer Software and Applications Confer. (COMPSAC), pp. 65. 2005.

[8] Cruz Filho, P.N. Teste de Software Baseado em Perturbação de Dados Dirigida por Padrões. In: Master Thesis, DInf-UFPR, June 2007 (in Portuguese).

[9] Heckel, R.; Lohmann, M. Towards Contract-based Testing of Web Services. Electronics Notes in Theoretical Computer Science, v. 82(6), 2004.

[10] Offut, J.; Wuzhi, X. Generating Test Cases for Web Services Using Data Perturbation. In: Workshop on Testing, Analysis and Verification of Web Services. July 2004.

[11] Tsai, W. T.; Paul, R.; Song, W.; Cao, Z. Coyote: An XML-based framework for Web services testing. In: 7th IEEE Intern. Symposium on High Assurance Systems Engineering. October 2002.

[12] Tsai, W-T; Wei, X.; Chen, Y.; Paul, R.; Xiao, B. Swiss Cheese Test Generation for Web Services Testing. IEICE Transactions on Information & Systems, v. 88(12). December 2005.

[13] Wuzhi, X.; Offut, J.; Luo, J. Testing Web Services by XML Perturbation. In: IEEE Intern. Symposium on Software Reliability Engineering. November 2005.

[14] W3C. Extensible Markup Language (XML) 1.0. http://www.w3.org/XML. October 2000.

[15] W3C Recommendation, SOAP Specification, June 2003. http://www.w3.org/TR/soap/.

[16] W3C. Web Services Description Language (WSDL) - W3C Recommendation, May 2001.http://www.w3.org/TR/wsdl.

Translating OWL Specified Domain Knowledge to Aspect Oriented Model

Juanzi Li, Xinyu You, Xiaoying Bai
Department of Computer Science and Technology, Tsinghua University
Beijing, 100084, China
{ljz,yxy}@keg.cs.tsinghua.edu.cn, baixy@tsinghua.edu.cn

Abstract- *OWL is a good candidate to establish the platform-independent machine-interpretable model of domain knowledge in an easy-to-understand, easy-to-use approach. It can also provide strong support for automatic model transformation and inconsistence checking by ontology reasoning mechanisms. Much research has been investigated to integrate ontology model with object-oriented models. However, current transformation methods suffer from the problems including code scattering, tightly coupling between classes and missing of structural and semantic information. To address the problems, the paper proposed an aspect-based approach to translate OWL ontology to AspectJ model. In this way, the knowledge can be separated from the code in the modelling phase and then integrated with the code automatically in the implementation phase. With the support of ontology technique, the OWL-specified domain model can be easily verified, reused and adaptive to changes. The proposed approach enables the externalization and automatic integration of the domain knowledge in the traditional object-oriented programming.*

1. Introduction

Inaccurate requirement understanding, complex software design and difficult code reuse are the important problems in software development. Various methods have been proposed to solve these problems, and model driven architecture (MDA) is an effective one. MDA aims at describing requirement using a platform-independent model and then translating the model to the implementation of the system [1]. The models in MDA include static structure model and event workflow, and static structure model is used more frequently in practical programming. How to describe the static structure model properly and how to adapt the change of static structure model in the implementation of the system are two problems in software development. We found that OWL ontology [2] is a good candidate to do this work. Thanks to its capacities of describing the domain knowledge preciously, OWL ontology is easy to understand and easy to use. At first, OWL ontology can bridge the gap between the requirement and design in software development. Secondly, OWL ontology can promote the software reusability because OWL itself is an open and platform independent mark-up language. At last, OWL can promote the automatic software development in model classification and inconsistence checking by using inference for OWL ontology.

At present, there exists a reasonable research to map OWL ontology to software programming languages such as Java. Most of the work focused on mapping OWL into object oriented (OO) model. Because of the differences between description logic and OO system, these methods have some problems such as meaning missing and structure inconsistent.

This paper provides a mechanism for mapping OWL ontology into aspect oriented model. By this mapping strategy, we can solve problems that existed in mapping OWL to OO model, and we have finished the translation tool based on it. The proposed approach enables the externalization and automatic integration of the domain knowledge in the traditional object-oriented programming.

2. Related Work

For the similar concept and relation, most research work was focused on mapping OWL ontology to OO model. In model translation, OWL ontology is translated some kinds of OO model language such as Java. Such good examples are Kazuki [3] proposed by David Rager, RDFReactor [4] proposed by Benjamin Heitmann, and Jastor [5] proposed by Ben Szekely.

Currently, all proposed methods focused on translating OWL ontology into OO model. As [6-7] stated: there exist fundamental differences in understanding OWL and OO system. For example, there is no class description in OO model. Especially, property in OWL ontology is independent of class, while the property in OO belongs to a part of class. Some problems of mapping OWL ontology into OO model are code scattering, tightly coupling between classes, and Code concentration.

A. Code Scattering.

As shown in Figure 1, in the OWL ontology, *Property1-3* are three resources independent of two classes *A* and *B*. When we map this ontology into OO model, the code of *Property2* will be scattered into *Class A* and *Class B*. This results in the problem of code maintenance. That is, if

Property2's description is changed, we need to change the code in both two classes synchronously.

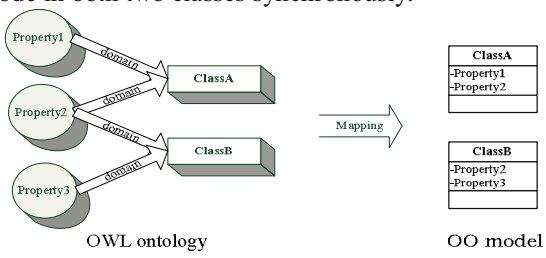


Fig. 1 Mapping OWL property to OO model

B. Tightly Coupling between Classes

In order to mapping the relationships between classes, translating model will obtain a lot of coupling relationships. As shown in Figure 2, to express the semantic of equivalent relationship between *ClassA*, *ClassB* and *ClassC*, six relationships in OO model are generated which make the coupling relationship are rather complex.

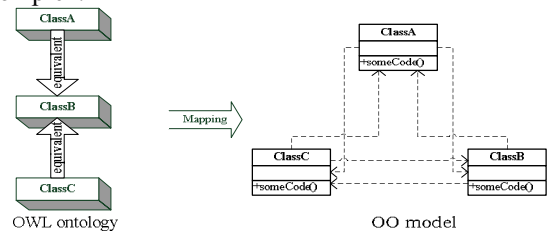


Fig. 2 Mapping equivalentClass to OO model

C. the Missing of Structure Information

Because OWL ontology and OO model have fundamental differences in structure description, translated OO model can not reflect the structure of OWL ontology. It makes the mapped model difficult to be understood. As shown in Figure 3, the property description and class description which are represented clearly in the OWL ontology are mixed together in OO model after translation as described in the right of Figure 3.

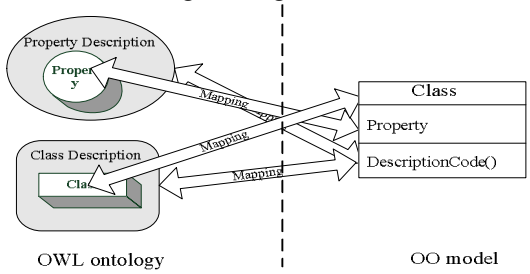


Fig. 3 Structure mapping between OWL ontology and OO model

As stated above, these addressed three challenges in model translation and also are the main concerns of this paper.

3. Mapping Principals from OWL Ontology to AO Model

As we know, there are two reasons which cause these problems. One is different structure between the OWL ontology and the OO model, and the other is the difference in description capacity of these two models language. To solve these problems, we try to find another programming model which is more flexible than OO model, and which is more suitable for the OWL ontology model.

In recent years, aspect oriented (AO) programming are becoming widely used to overcome the limitations of OO programming model [8]. AO programming introduces a new modular unit, called "aspect", for the specification of crosscutting concerns. It organizes the crosscutting relationships between objects by using weaving mechanism so that it can decoupling the relationship between objects and help to concentrate software's code. Aspect oriented model can express the syntactic and semantic features of the OWL ontology rather properly. By taking advantage of "aspect" module unit and its flexible crosscutting weaving mechanism, the problems addressed in translating the OWL ontology into OO model translation can be properly solved.

A. Code Concentration

AO model can encapsulate OWL elements including both class and property while keep their independence of each other. As shown in Figure 4, the properties in the OWL ontology can be mapped into AO model independently, and also these property aspects can be weaved into corresponding classes by inter-type declarations. It makes the codes be able to concentrate on the separate properties aspects.

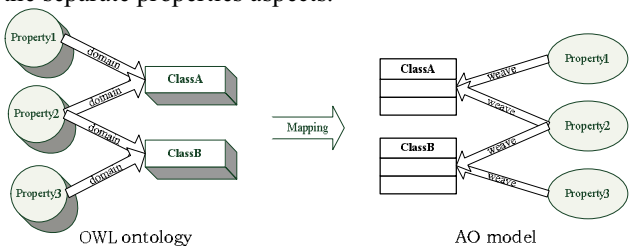


Fig. 4 Mapping OWL property to AO model

B. Decoupling Class Relationships

In AO model, we can generate an independent aspect for the relationships between multiple classes. As shown in Figure 5, *AspectEquivalent* is an aspect to show the equivalent relationship, the equivalent relationships between three classes *A, B* and *C* can be obtained by weaving *AspectEquivalent* into these classes respectively and only three relationships are needed.

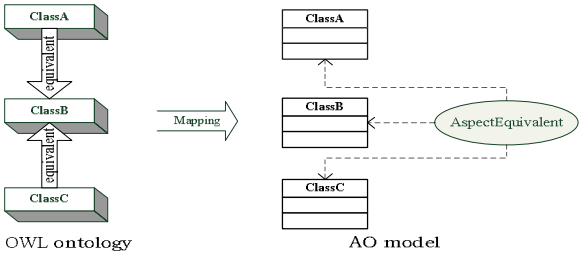


Fig.5 Mapping equivalentClass to AO model

C. Structure Consistency

The model generated by AOP is finer-grained and flexible, and it keeps the model structure consistency between the OWL ontology and the translated AO model. This makes the transformed model easily understandable, and also the transformed codes are easy to be maintained and reused. In our proposed method, three kinds of elements including class, class description and property in the OWL ontology are mapped to class, class description and property aspect respectively in the AO model. As shown in Figure 6, compared with mapping into OO model, mapping into AO model can keep the structure of the OWL ontology very well.

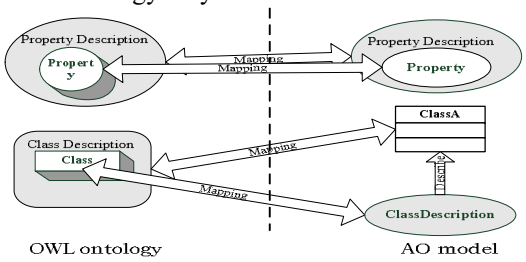


Fig. 6 Structure mapping between OWL ontology and OO model

In the following, we will propose a comprehensive translation method to map the OWL ontology into a general AO model using AspectJ [9]. Two principals are used throughout the translation. For the part of the OWL ontology, which is consistent with the OO model, we use the mapping method used in the mapping from the OWL ontology into the OO model. For the other part which are inconsistent with OO model in the OWL ontology such as class description, properties and restriction, we propose to aspects and weaving mechanism to perform the translation.

4. The Mapping from OWL Ontology into AO Model

In the translation, we first define interface IThing to be the ancestor of all mapped classes, it corresponds to class Thing in the OWL ontology. Each generated class has a unique id property which is set in the class constructor corresponding to the class name in OWL ontology. In the following, we will give the translation methods in detail.

A. OWL Class and Class Description

Referring to the class mapping method proposed by Aditya Kalyanpur [10], we make use of the well matching between the OWL class and the AspectJ code to perform the class mapping as follows. Each OWL class is mapped into an interface and an implementation class. The class description and class's restrictions on properties are mapped into a class description aspect in AspectJ.

1) Class

A class in the OWL ontology is mapped into an interface and implementation class in AspectJ. For example, class *A* in the OWL ontology will be mapped into interface *IA* and interface implementation class *CA* in AspectJ.

2) subClassOf

A subClassOf *B* in OWL is mapped into a inherit relationship between interfaces of two mapped classes *IA* and *IB* in AspectJ.

3) equivalentClass

Let OWL the equivalent class map into the interface with the same properties, and each equivalent relationship between classes is mapped into an equivalent relationship aspect. In this aspect, we declare these equivalent classes to implement all these equivalent interfaces. For example, *A* and *B* are two equivalent classes, in the equivalent relationship aspect, we create following AspectJ codes:

```
declare parents:(A||B) implements IA,IB;
```

Based on the interface programming, class *A* and class *B* can be used alternately so that the semantic of equivalent between class *A* and *B* can be expressed.

4) intersectionOf

In the OWL ontology, the *A* intersectionOf (*B*, *C*) represents that class *A* has the characteristic which both class *B* and class *C* has. We can translate it to this code in class description aspect:

```
declare parents : IA extends IB, IC;
```

In this way, *IA* will get the all characteristic of *IB* and *IC*.

5) unionOf

In the OWL ontology, unionOf has the opposite meaning with intersectionOf. We use this code to express it:

```
declare parents: (IB || IC) extends IA;
```

6) disjointWith

In the class description aspect, we can define a function with the same function name which returns a different type for all the member interfaces of this relationship. It can avoid these interfaces being extended or implemented by a same interface or class. So it can express the semantic of disjoint relationship in OWL ontology.

7) complementOf

A complementOf *B* in OWL represents two complement classes. In the class description aspect, let *IA* and *IB* are

two interfaces extend from the most top interface *IThing*. Then, we use the mapping method of disjointWith to make them be disjoint with each other, at last we can use class pattern expressions to let all interfaces which do not extend from *IB* extend from *IA*. So the mapping can be described as:

```
public IA IA.forDisjoint (){…}
public IB IB.forDisjoint (){…}
declare parents:(!IThing && IThing+ &&(!(IB
+))) extends IA;
```

8) oneOf

We check the value of object's id property in its constructor. If the id does not belong to the id set which is defined in the OWL oneOf statement, an exception would be thrown out.

9) Cardinality

We set a pointcut for the property setter in the class description aspect, and define an advice to check the number of items in the list of property values before this pointcut, if the number of items is not consistent with the cardinality, an exception is generated. In this way, we can separate the property restriction from the property definition and we can define different property restrictions for the same property in different classes. For example, in the OWL ontology, the cardinality of property *P* restricted on class *A* is 1, it will map into the AspectJ codes in class description aspect which is described as:

```
pointcut setP(List arg):set(List IA.P)&&
args(arg);
before (List arg):setP(arg){
     if(arg.size()!=1)
     throw new CardinalityException(…);}
```

10) hasValue

In the same way, we can use the similar mapping method with the cardinality mapping to check whether the list of the property values contains the object with correct object id.

11) someValuesFrom/allValuesFrom

We use the similar methods with the cardinality mapping to check if the objects in the list of the property values belong to right classes.

B. OWL Properties

OWL properties consist of the description of property constructs, the characteristics of properties and relations to other properties. Following gives the mapping from the property descriptions in the OWL ontology to AspectJ

1) domain

Domain describes the classes which have the properties. It defines which classes could have the properties. In the mapping into property aspect, the domain class points out a target classes set which the property to be weaved into. Firstly, we define a domain interface which represents all the domain classes. Secondly, we add the property to the domain interface by using inter-type declares. Finally, let the interfaces in the domain extend from the domain interface.

For example, person and publisher are two domain classes of property hasContact. In the property aspect, we define:

```
private List IhasContactDomain.hasContact;
declare parents:(IPerson||IPublisher) extends
IhasContactDomain;
```

If the classes in domain changed in the OWL ontology, we only need to modify the weaving target set (*IPerson*||*IPublisher*).

2) range

Range is used to describe the data type of property's value which a property can take. We use a list to represent all the property values in AspectJ, and check the type of all items in this list when setting the value of property.

For example, property hasContact has the range of class Contact, it can be mapped into:

```
public void IhasContactDomain.setHasContact
(List values){
…//check the values listed in the list, if the
value is not the types defined in the list, an
exception will be thrown out.
this.hasContact = values;}
```

3) Functional

Functional property is a property that can have only one (unique) value *y* for each instance *x*, i.e. there cannot be two distinct values *y1* and *y2* such that the pairs (*x*,*y1*) and (*x*,*y2*) are both instances of this property. In the mapping, when setting the property value, we can check the number of items in the list of property values. If the number is greater than 1, then an exception is thrown out.

4) InverseFunctional

InverseFunctional property means that this property value is unique in this system. In the property aspect, we maintain a property value list for all the Objects which have this property, and check the value when the property value is set. If the value has been existed, an exception is thrown out, else set the value and add it to the list.

5) Symmetric

A symmetric property is a property for which holds that if the pair (*x,y*) is an instance of P, then the pair (*y,x*) is also an instance of *P*. In the mapping, we use *P*(*a,b*) to indicate that object *a* has property *P* with value *b*. Symmetric property *P* (*a,b*) means *P*(*b,a*). In the property aspect, we add code to make *b.P=a* after *a.P* is set as *b*.

6) Transitive

The transitive property *P* means that *P(a,b)*, *P(b,c)=>P(a,c)*. To describe its semantic, we add a function to return all the transitive values. It means, for object *a*, the function will return *b* and *c*. For *b*, the function will return *c*.

From above description, for an OWL ontology described in OWL-DL, we can translate it into AspectJ codes which maintain the semantics of the ontology properly.

5. Implementation and Experiment

Based on the mapping principals and mapping rules described in section III and IV, we implemented a model translation tool which named OWL2AspectJ to translate the OWL ontology into AspectJ code. The input is a valid OWL document and the output is its corresponding AspectJ codes.

We use JDK1.5 as the running environment and use protégé OWL API as the OWL parser. For an OWL document input, the translation tool first parses the document using the parser and then maps it into AspectJ code automatically according to the mapping strategy proposed in the paper.

Using the OWL ontology presented in http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine# as an example, after the translation, we can get the AspectJ code as shown in Figure 7:

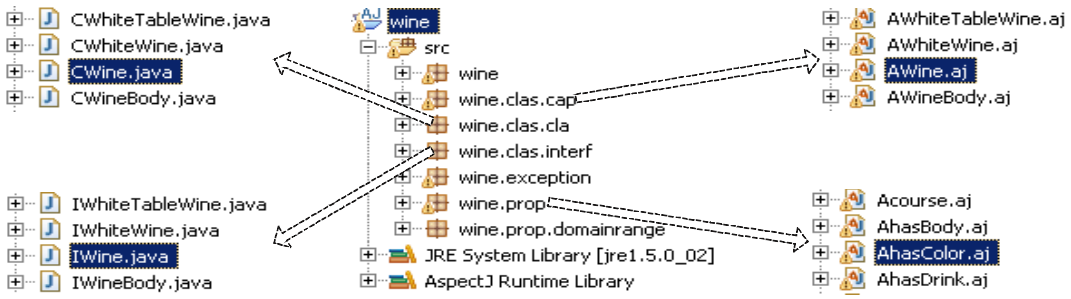


Fig. 7 Translated AspectJ codes structure

Where, *win.clas* contains all codes generated from classes and class descriptions in wine ontology. *clas.interf* contains the interfaces of the classes. *clas.cla* defines all the implementation classes. *clas.cap* contains the class description aspects. In the property mapping, *wine.prop* contains the codes generated from properties and their characteristics in OWL ontology. Where, *wine.prop* is the aspect of properties, and *wine.prop.domainrange* corresponds to the interfaces of domain and range. These codes are independent of each other, the aspects of classes and properties are interacted through the AspectJ weaving mechanism.

For the classes in ontology wine, after translation, three independent parts of codes are generated. They are the class interface *wine.clas.interf.IWine*, interface implementation class *wine.clas.cla.CWine* and class description aspect *wine.clas.cap.AWine*. The most important codes are defined in the part of *AWine*, it can operate on *IWine* by weaving mechanism to keep the semantic of the ontology. For the properties in Wine ontology such as *hasColor*, after translation, property aspect *wine.prop*, the interface of property *domain wine.prop.domainrange*, the interface of the property range *wine.prop.domainrange*, and *IhasColorRange* are generated respectively. The main codes are in *AhasColor*, and it can be weaved into corresponding classes.

6. Evaluation of the Mapping

In section IV, we have described the mapping method for the classes and properties in the OWL ontology in detail. Here, we use two examples to test whether the transformed codes can express the semantic presented in the OWL ontology.

Example 1: *ConsumableThing* and *NonConsumableThing* are two complement classes, we use following program to test the generated codes.

```
IWine w=new CWine();
if(w instanceof IConsumableThing)
System.out.println("w is ConsumableThing");
if(w instanceof INonConsumableThing)
System.out.println("w is NonConsumableThing");
IRegion r=new CRegion();
if(r instanceof IConsumableThing)
System.out.println("r is ConsumableThing");
if(r instanceof INonConsumableThing)
System.out.println("r is NonConsumableThing");
```

We could get the following result:

```
w is ConsumableThing
r is NonConsumableThing
```

This test indicates that the generated codes can classify wine and region into *ConsumableThing* and *NonConsumableThing* correctly.

Example 2: We also test the class description on oneOf for class *WineColor* by the following code:

```
IWineColor wc=new CWineColor("White");
```

It runs correctly. Then we run :

```
IWineColor wc=new CWineColor("blue");
```

The program throws an exception:

```
wine.exception.OneOfException: blue is not one
of {White Rose Red}
```

It indicates that generated codes can use the exception processing to constrain the semantic of oneOf.

7. Benefits from the Model Translation

The OWL ontology has high powerful expressive capacity and the AO model has high flexibility in programming. Using the translation from the OWL ontology into AO programming model may get following advantages.

A. Readability and Maintainability of Translated Code

How to keep the consistency between document and code has been a challenge in software development. The OWL ontology is build based on the concepts and relationships of a specific domain, and it can be easily understood by human and at the same time currently we have some OWL ontology editor tools with friendly user interface. On the other hand, the AO model is high consistent with OWL ontology in structure description. So we can use OWL ontology as the document of code and furthermore we can read, maintain and reuse the codes by using the OWL document. Once the OWL ontology is changed, software developers can quickly locate the code to be modified instead of calling translation again which can avoid overwriting the previous information.

B. New Programming Model Benefiting from the Code Structure of Property Independence

In the traditional OO programming model, property is treated as a class element and it is tightly coupled with one class. In fact, one property can exist in multiple classes. In this case, when a property changes, we have to modify the class codes with which each property is connected. It may result in the problem of code maintenance. Using the mapping method proposed in this paper, we can constrain the modification in the property itself. Also, if different classes have different requirements for this property, we can define the property restrictions in the classes.

As we know, in the translated AspectJ codes, each property has an interface to present its domain. In the development, we can deal with this property instead of considering the objects related to the property so that it provides a flexible programming model. For example, a cart is represented as a list of commodities object with different class. If we want to get the total price of all these commodities in the cart, by using traditional method, we need to know all items class before adding up their price. Now, we can only process the price's property independently and do not need to consider their real classes. What we should do is to cumulate the price by seeing all the items as the instances of *IPriceDomain*.

C. Rich Semantics of OWL Ontology would Bring New Characteristics to Programming

Translating the OWL ontology into the AspectJ code can maintain the semantics of the ontology and bring out some new characteristics for programming model. For example, in a marketing system, all classes are classified into two complement classes, *consumable* thing and inconsumable thing, and each has different characteristics. If we add a new inconsumable class *A* into the system and class *A* did not extends from any inconsumable class, class *A* will be seen as an inconsumable thing and get all characteristics of inconsumable thing automatically.

8. Conclusions

This paper proposed a new method to mapping OWL ontology to AspectJ. The methods can translate most elements in OWL DL into AspectJ code, and the translated codes can express the semantic of OWL ontology and is high consistent with the OWL ontology in both structural and semantic description. With the development of software and knowledge engineering, MDA based software development and ontology based knowledge management have been widely used. The convergence of these two fields shows a promising trend in software development. OWL ontology gives the well defined semantic model in the requirement domain, in the software development, developers can extract the OWL ontology they are concerned with. Then, the translation from the ontology to AspectJ can be performed to generate the corresponding aspect model to be the static structure of the system. At last, the further development can be made. In this way, we could speed up the software development and reduce the development investment.

References

[1] A. G. Kleppe, J. B. Warmer and W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise, Addison-Wesley Professional, 2003.

[2] M. K. Smith, C. Welty and D. L. McGuinness, "OWL Web Ontology Language Guide," W3C Recommendation, vol.10, 2004.

[3] Kazuki: http://projects.semwebcentral.org/projects/kazuki/

[4] RDFReactor :http://ontoware.org/projects/rdfreactor/

[5] Jastor :http://jastor.sourceforge.net/

[6] H. Knublauch, D. Oberle, P. Tetlow and E. Wallace, "A semantic web primer for object-oriented software developers," in Proc. 2006 World Wide Web Consortium.

[7] IBM Sandpiper Software, Ontology Definition Metamodel, 2005

[8] J. D. Gradecki and N. Lesiecki, Mastering AspectJ: Aspect-Oriented Programming in Java, John Wiley & Sons, Inc. New York, NY, USA, 2003.

[9] R. Miles, AspectJ Cookbook, O'Reilly Media, Inc., 2004.

[10] A. Kalyanpur, S. Battle and J. Padget, "Automatic mapping of OWL ontologies into Java," in Proc. 2004 Software Engineering and Knowledge Engineering.

MAPLE: a Maintenance Approach for Pattern-enabLed rEconfiguration of SOA-based enterprise application

Songlin Hu, Ying Liang, Jiuming Tian, Yicheng Song

***Abstract*--Service-Oriented Architecture (SOA) is widely adopted as a way to reach flexible integration of enterprise system, and Event Driven Architecture (EDA), which enables complex event processing as well as asynchronized communication among systems, is also attracting more attentions. The deployment of such an environment makes it possible for organization to improve interoperability and agility, and brings about new challenges to the maintenance and evaluation of the whole system at the same time. The paper first analyzes problems retrieved from a real engineering scenario, and proposes the MAPLE approach to implement two-level abstraction by reusing and reconfiguring of business and technical patterns. As showed by a legacy system integration experience in a Textile and clothing manufacture company, it provides a practical route to achieving maintenance and evolution of the EDSOA (Event Driven SOA)-based enterprise application.**

***Index Terms*--Maintenance and evolution; SOA based system; Event driven; Business pattern; Enterprise integration pattern; Textile and clothing manufacture.**

I. INTRODUCTION

As a popular paradigm, SOA is becoming a hot topic both in industry and academy. It shows some unachievable high level advantages when combined with event technologies, and is now treated as a fairly good choice for building large scale systems as well as reuse and integration of legacy systems.

Topics in early stage of the whole lifecycle of conducting a SOA based system have been the focuses of works in service computing area. The maintenance and evolution of such a system is becoming popular recently [1]-[2]-[3], but only a few current works have been presented to discuss it.

The objective of maintenance here is to modify the existing system while preserving its integrity. As a consequence, problems including expression of changes, impact analysis, change propagation, and revalidation should be tackled.

Even though the SOA based system, power by loose-coupled building block and flexible composition mechanism, has the potential ability to be adaptive to the changing requirements, it is still challengeable to have the complicate system reliably "reconfigured" according to new situations.

To achieving the goals mentioned above, the MAPLE is put forward here to describe and manage the relationships between technology and business models via two level abstractions, and enables maintenance by reuse and reconfiguration of the relationships. The objectives we are aiming at would be broken into two levels: the first one is BPEL reconfiguration in response to changes in business model, the second one is high level service reconfiguration in response to changes of usage of certain resources.

The paper is organized as below: the first section gives an overview of related work, engineering background, and presents the challenges generated from real application scenarios. The MAPLE approach is introduced in Section 2, including conceptual idea of pattern based two-level abstraction. Section 4 shows the whole framework along with roles for it. In Section 5, our application experience in a Textile and clothing manufacture company is illustrated. At last, conclusions and future works are presented.

II. RELATED WORKS, BACKGROUND & MOTIVATION

A. At Process Level

Whether it is named as abstraction or not, abstraction of process has been studied to improve the capability of process modeling and execution. Recently, semantic web service composition has been focus of famous SOA research groups, like Meteor-s [5] and SWARD [6], which can achieve semantic abstraction of services, and facilitate dynamic service selection at run time. While at the same time, model-driven BPEL management creates another way to process abstraction. IBM has put forward a product to map UML model to BPEL [7], allowing process modeler to create UML

This work was supported in part by the Chinese National High-Tech. R&D Programa under Grant 2006AA04Z158 and 2006AA01A106, and the National Basic Research Program, China, under Grant 2007CB310805 and 2005CB321807.

Songlin Hue is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China (e-mail: husonglin@ict.ac.cn).

Ying Liang is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China (e-mail: liangy@ict.ac.cn).

Jiuming Tian is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100910, China. He is also with Graduate University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: tianjiuming@ict.ac.cn)

Yicheng, Song is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100910, China. He is also with Graduate University of Chinese Academy of Sciences, Beijing 100049, China. (e-mail: songyicheng@software.ict.ac.cn).

at first, and then map the UML models to corresponding BPEL. Similarly, Aalster proposes an approach and related algorithms to translate Petri-net to BPEL model [8]. The process modeler who is familiar with Petri-net-Like workflow can indirectly build BPEL without know the technical detail of BPEL languages.

In practice, the logic of BPEL is always much too complicated than the logic of business level processes, especially for the processes with asynchronized events as well as service invocations. With event driven features, the BPEL implementation of the business logic has to be fulfilled by using “receive”, “onMessage”, ”Correction” etc., which make the diagraph of the business process “stretched” or totally changed. The logic of BPEL is also affected by available services. Moreover, the users who want to use model driven BPEL approach should have to be familiar with technical models like UML, which is also hard for business user to grasp. Obviously, it is necessary to find a new way to enable the IT manager in the serving stage to adjust the processes to meet new requirements.

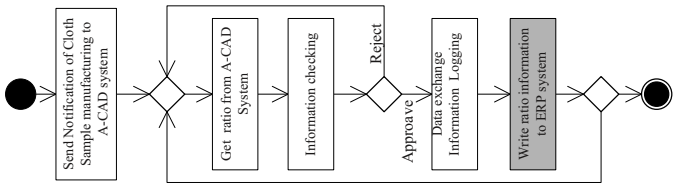


Fig. 1. Process model from business point of view

Fig 1 illustrates a real business level process. It describes the business logic for a data exchange procedure between legacy ERP and CAD systems. At the beginning, a notification is sent to a CAD workstation with initial data as soon as the ERP system receives a new order, and the design result will be exchanged back to the ERP system after the CAD designer finishing the task assigned. During the exchange procedure, a manager of the ERP system will verify the data, check to see whether to accept the data or not. If the data is rejected, the CAD designer will get the feedback, and executing the task again.

The process digraph looks sample from business point of view, while the corresponding BPEL model has to be designed as the one showed in Fig 2. It is obviously that it is far from practice to map the business level process like Fig 1 to an executable BPEL by using formalized specifications and automatic algorithms. It is essential to find another way to implement the abstraction.

In practice, due to the management standards or routines, organization structure and the functionalities assigned to the departments, there exist lots of similarities among business logics. The idea here is to take advantage of the patterns to construct the relationship between business view and technical view of the process, and to facilitate the design or modification of processes by reusing or reconfiguring the patterns. With the support of patterns, maintenance of BPEL can be achieved by reconfiguring high level process, while detecting dependency and impact of BPEL could be executed automatically, and the changing of BPEL could be implemented with automatic algorithm and sometimes with user interaction if needed.

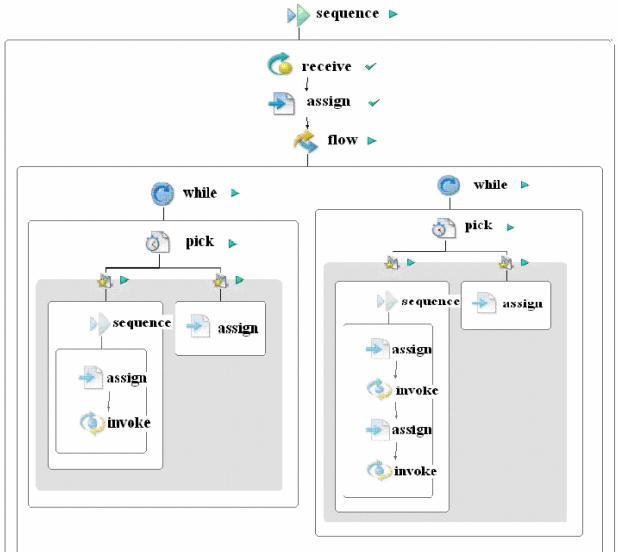


Fig. 2. BPEL model for Fig 1.

B. At Service Level

At service Level, wrapping of legacy system to web services can be treated as a kind of abstraction [9]. A service in one process can also be a set of abstract interfaces for another process. Abstraction can be semantic description of service interfaces using Meta-data or ontology as well. In this paper, we focus on high level abstraction of service by using Enterprise Integration Pattern (EIP) [10]-[11].

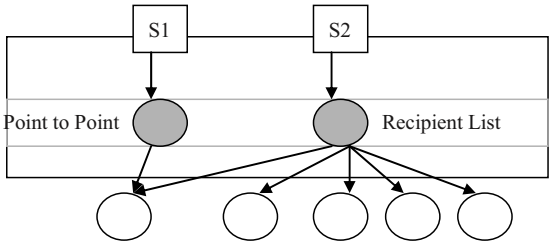


Fig. 3. Service Level Abstraction using EIP

Asset in an enterprise can be utilized in different way at different time. For example, in our engineering project, information A generated from ERP system is sent by process 1 to one CAD workstation. After a period of time, it is asked to be sent to a design group that contains a number of CAD workstations. Here, we can define one services S1, and generate S2 just by changing the integration pattern it uses.

The integration style for both of them is messaging. The Messaging System for S1 is message channel (point to point), while S2 should adopt message router (Recipient List).

Patterns like Translator and Message router can also be used to enable data transformation and complex message Routing. In this way, we can define services for different usage, which can enable service level abstraction.

III. MAPLE

Fig 4 gives a conceptual view for two-level abstraction. Process level abstraction is to build abstract business level process based on business pattern, while EIP based abstraction is a kind of service abstraction, which constructs abstract service using integration pattern.

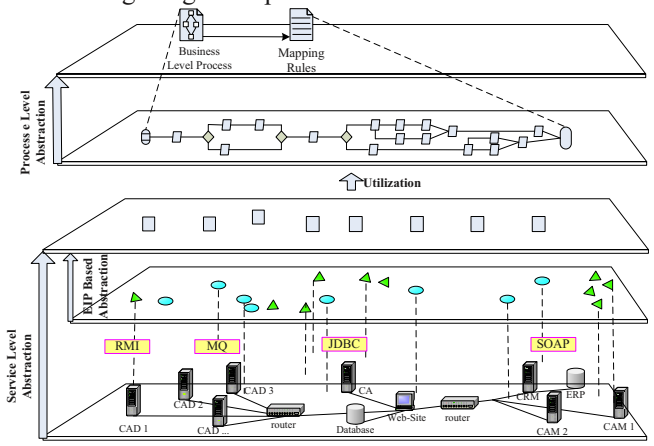


Fig. 4. Two-Level Abstraction

A. Business Template

In this approach, we use business template to describe business level process, BPEL, and the relationships between these two to represent a particular kind of business pattern. It is defined as following:

< business template >::= <ID>< business process >< BPEL Template >< Mapping Rules >
<business processes> ::= < service component >< logic node >
<service component> ::= < Messaging component >< RPC component >
<logic node>::= <sequence activity ><pick activity >< flow activity >< while activity >< start activity >< end activity >
*< BPEL Template > ::= < BPEL File >{< WSDL Files >}**
*< Mapping Rules >::= {< service component >< BPEL Object >}**

Mapping rules are the relationships between Service Components in business level process and Objects in BPEL, it only contain elements that users can modify without interfering with the whole process logic for a particular pattern. So far, 4 BPEL objects, namely: Invoke, receive, onMessage and reply activity, are involved in our approach. The validity of mapping rules should be guaranteed by template designers.

Service component mapping is mainly related to several elements in the BPEL and WSDL files, the specific relationship are listed as follows:

RPC component is related to “Invoke”, while the Message component could be “Receive”, “onMessage”, or “reply”. Any change to these two components will affect some other relevant elements besides themselves.

The operations allowed in a business pattern are:

Component changing: the operation is to change a component in business level process. It will result in change of related BPEL object. The system will synchronize reconfigure relevant elements, and will ask for user interaction to edit “Assign” objects if needed.

Component adding: the operation is to add a sequential or parallel component to the business level process. Corresponding object and relevant elements will be added to the BPEL.

Component deletion: the operation is to delete a component if it has no relationships with other components. Corresponding object and relevant elements will be deleted

Assign Modification: the operation is to modify the variables and their relations in “Assign” object. It is utilized to guarantee the validation of data flow.

The overall procedure for reuse of one pattern contains four steps: 1) Reconfigure the Pattern by an operation; 2) Automatically Detect impacts; 3) Read parameters and messages information from WSDL file for the service being change, and automatic change the partial structure of the BPEL, d; 4) Ask for user interaction to modify BEPL “Assign” object for data flow if needed.

TABLE I
RELATED ELEMENTS THAT WILL BE AFFECT BY CHANING OF PARTICULAR KIND OF COMPONENT

Component / Elements	RPC	Messag
WSDL file	Y	
import	Y	
partnerlink	Y	
partnerlinktype	Y	
variable	Y	Y
assign	Y	Y

B. Enterprise Integration Pattern

We propose abstract service based on Enterprise Integration Pattern. An abstract service shows high level view of one operation of service to users, from which the users can get the information of how the service works. It describes the EIP used by the services, corresponding end points and pattern dependent components it will use. The definition of Abstract Service is described as below:
<Abstract Service>::=<ID><Operation><WSDL><EIP pattern><End points><Pattern dependent Components>

Pattern dependent Components is the component related to particular pattern, like Message Router component or Message Translator component.

The abstract service can be directly mapped to the deployment files of the end points in the Enterprise Service Bus product like ServiceMix.

IV. FRAMEWORK AND ROLES

A. MAPLE Framework

The framework for MAPLE is illustrated in Figure 5. Besides tools that are already provided by current venders, like BPEL tools, ESB, Portal and IDE, four modules are added to facilitate the MAPLE approach, including:

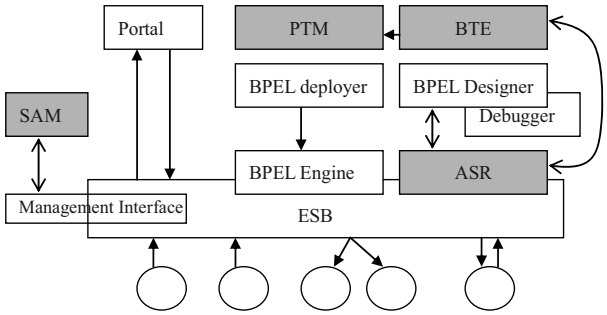


Fig 5. MAPLE Framework

Service Abstraction Manager (SAM): the SAM is the toolkit for designing abstract services using different enterprise Integration pattern. The configuration information of abstract services will be restored in Abstract Service Repository.

Abstract Services Repository (ASR): Repository for abstract services. BPEL designer and BTE will use it to get service and related WSDL file. It also maintains abstraction information that contains relationship among service and the endpoints related to it according to certain EIP. It can be used by BTE and BPEL designer.

Business Template Editor (BTE): It is utilized to model business level processes, and to edit mapping relationships between business level processes and BPEL. Changes of BPEL can be achieved by configuring the template, and the creation of new process can be done in the same way to generate new template with modified business level process and BPEL. The service components can be selected from ASR.

Business Template Mapper (BTM): Mapping changes of business level process to BPEL, detect impacts by rule and modify related objects in response to the changes.

The first two are used to manage service level abstraction, while the others are developed for process level abstraction.

B. Roles for the framework

Mattsson[12] has presented preliminary ideas on the roles required for developing, evolving and maintaining SOA-based systems. The roles are categorized into 7 groups, namely: SOA Front end support, Back end support, Traditional Back end support, SOA design, SOA management, Quality Assurance, Business Project Members[]. Here, we focus on roles for the maintenance & evolution system, especially the roles closely related to the Framework, while ignoring the others. The roles are:

Interface developer: The role belongs to *Traditional Back end support group*, and is responsible for developing and verifying interfaces based on legacy systems to meet the requirement of Service Abstraction Manager. It works in initial and evolution stages.

ESB Component developer: Responsible for designing, developing, testing of additional building blocks for integration and management, like logging service, auditing services, etc. The Service Components are plugged into Enterprise Service Bus as well. It works in initial and evolution stages.

Service Abstraction Manager: Responsible for managing original service end points and Service components, as well as data transformation rules deployed to ESB. The role creates abstract services by using suitable Enterprise Integration Pattern, and maintains the routing configuration in response to changing requirements of services. It can be treated as the combination of *Service Developer in SOA Back-End Support group* and *Service Designer in SOA Business Process Design group*. It works in initial, evolution and serving stages.

Business Process Orchestrator: The role develops BPEL based on the business process provided by Business Process Architect. It belongs to *SOA Business Process Design Group.* It works in initial and evolution stages.

Business Template designer: Designs business templates according to Business patterns. It has the responsibilities of both *Business Process Manager in SOA Back-End Support group* and SOA *process manager in SOA Management group.* It works in evolution and serving stages.

Process maintenance manager: The role utilizes the template defined by Business Template designer to construct new BPEL or to reconfigure the process, so as to implement small changes. It is similar as *Business Process Manager in SOA Back-End Support group.* It works in serving stage, and can also be used to rapidly create processes in evolution stage.

Interface developer and ESB component developer uses IDE like eclipse, Business Process Orchestrator uses BPEL designer, while the others will use the new modules in figure 5. SAM and SAR are tools for Service Abstraction n manager, Business Template designer and Process maintenance manager will use BTE and ASR.

V. CASE STUDY

The approach proposed in this paper has been applied in an ongoing integration project for a large textile and clothing Manufacture Company. The company owns 23 categories of IT systems like: D/CAM/CIM, PDM, ERP, etc. The integration project was launched to cope with the problem mentioned above in early 2007. After requirement analysis, a high level business process is draw to illustrate the overall collaboration process for order-driven manufacture.

By analyzing the business process, we find that the atomic process of the enterprise can be separated into several categories: The first category involves data exchange of fabric rate, fabric width, size scale, operation schedule, etc.; the second category involves management of the samples, files, production schedules etc.; the third category involves verification of cutting consumption, patch control etc. Each category contains large number of atomic service. Obviously, it will take great efforts to implement realization and

maintenance all of these processes. This paper proposes a new method which simplifies the creation and adjustment of processes by means of classifying the business processes with the similar logic, refining the business patterns, designing and generating business templates.

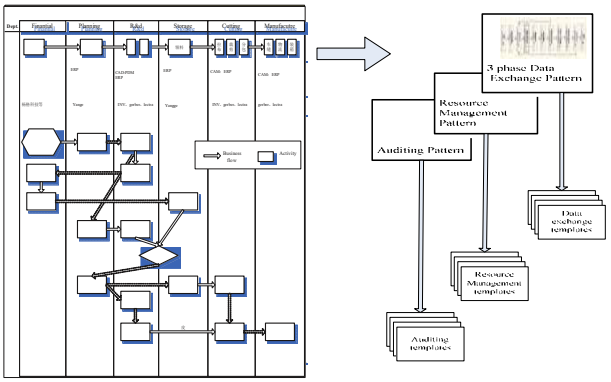


Fig. 6. Overall procedure and related business Patterns

Fig.7 shows an example of event-driven data exchange pattern, which includes 8 steps. The BPEL implementation related to this process has the same structure with the BPEL showed in figure 2.Figure 8 shows some parts of the BPEL framework in the basic template.

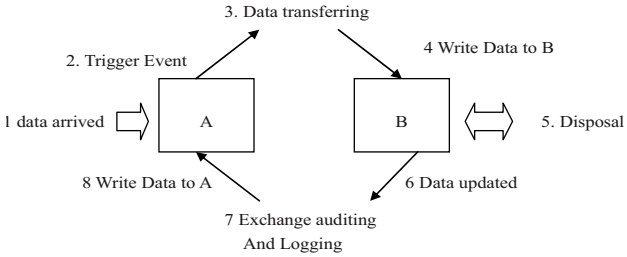


Fig. 7. The procedure for Event driven Data Exchange pattern

```
……
<bpel:process……xmlns:ns1=”http://ict.ac.cn”
      xmlns:ns2=”http://ict.ac.cn/xsd” ……>
<bpel:import…… location=ServiceLocation
      namespace="http://ict.ac.cn"/>
<bpel:import…… location=AServiceLocation
       namespace="http://ict.ac.cn/xsd"/>
……
<bpel:variable name="messageIn"
 messageType="ns2: WriteDatatoASoapIn" />
<bpel:variable name="messageOut"
 messageType="ns2: WriteDatatoASoapOut" />
……
<bpel:copy>
<bpel:from…… >……</bpel:from>
<bpel:to part="parameters" variable="MessageIn">
       <bpel:query>ns2:listType</bpel:query>
</bpel:to>
</bpel:copy>
……
<bpel:invoke inputVariable=" MessageIn "
      operation="WriteDatatoA" outputVariable="MessageOut"
      partnerLink="ERPPL"
      portType="ns2:ServiceSoap"/>
……
</bpel:process>
```

Fig. 8. A set of BPEL information for one invoke node filled with grey in Fig. 1. The words in italics are about the node.

```
……
<bpel:process……xmlns:ns1=”http://ict.ac.cn”
      xmlns:ns2=http://ict.ac.cn/xsd”
      xmlns:ns3="http://sizeratio.example.org"
      xmlns:ns4="http://erpservice.example.org/"
      xmlns:ns5="http://sizeratio.example.org/xsd" ……>
<bpel:import…namespace="http://sizeratio.example.org"
location=http://127.0.0.1:8080/axis2/services/SizeRatio?wsdl />
<bpel:import…namespace="http://erpservice.example.org"
location= "http://10.61.0.11/ERPWriter/?wsdl" />
……
<bpel:variable name="messageIn"
 messageType="ns4: WriteSizeRatioSoapIn" />
<bpel:variable name="messageOut"
 messageType="ns4: WriteSizeRatioSoapOut" />
……
<bpel:copy>
<bpel:from…… >……</bpel:from>
<bpel:to part="parameters" variable="MessageIn">
       <bpel:query>ns4:listType</bpel:query>
</bpel:to>
</bpel:copy>
……
<bpel:invoke inputVariable=" MessageIn "
      operation="WriteSizeRatio" outputVariable="MessageOut"
      partnerLink="ERPPL"
      portType="ns4:ServiceSoap"/>
……
</bpel:process>
```

Fig. 9. A set of BPEL information for a specific invoke node. It is generated by change the empty invoke node to an existing service. The Business Mapper will retrieve information from related WSDL file, and automatically modify the BPEL files.

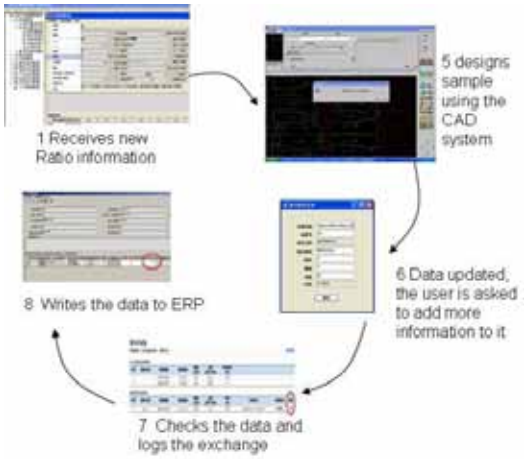


Fig. 10. The system user interfaces when run the BPEL to implement an event driven data exchange process between ERP and CAD systems.

The process showed in Fig 1. one can be generated from this template. Suppose ERP is treated as system A, and the user select a ERP write operation “*WriteSizeRatio*” in the service located in “*http://10.61.0.11/ERPWriter/?wsdl”* to take the place of the “*WriteDatatoA”* operation in the basic template. BTM will read information from the WSDL, and map the change to the BPEL. Figure 9 shows the generated template, with the operation replacement. In BPEL code, the words in italics are differences between basic template and generated template.

In this process, abstract services are configured using different pattern. For example, in step 2, event generated in the ERP system is wrapped to be a JMS message. Since the event should be used by another process, publish subscribe pattern is utilized, where the end point acts as publisher and the abstract service acts as subscriber.

After generating new BPEL for the ratio Data exchange between ERP and CAD systems, it can be deployed to the platform. Figure 10 illustrates some user interfaces.

VI. Conclusion

A maintenance approach for SOA based system is proposed based on two-level abstraction, which provides a practical way to facilitate reconfiguration of SOA processes and abstract services. Experiences accumulated in our first stage application in a real SOA based project prove that it can achieve easy-to-use reconfiguration without introducing unexpected effects to the whole system.

The future works includes improving the framework, implementing new functionalities for the new modules, formalized study on the descriptions of business template and abstract service, etc. Semi-automatic generating and validating of business pattern itself based on BPEL can be an interested topic as well.

VII. References

[1] Grace Lewis, Dennis Smith, Kostas Kontogiannis, Scott Tilley, Mira Kajko-Mattsson, Ned Chapin, "A Research Agenda for Maintenance & Evolution of SOA-Based Systems," IEEE International Conference on Software Maintenance, Oct. 2007, Paris, France, pp.481-484.

[2] Kostas Kontogiannis, Grace Lewis, Dennis Smith, Marin Litoiu, Hausi Muller, Stefan Schuster, Eleni Stroulia, "The Landscape of Service-Oriented Systems: A Research Perspective," International Workshop on Systems Development in SOA Environments, 2007. pp.1 – 1.

[3] Kajko-Mattsson, Mira Lewis, Grace A. Smith, Dennis B, "Evolution and Maintenance of SOA-Based Systems at SAS," Proceedings of the 41st Annual Hawaii International Conference on System Sciences, 2008, pp. 119-119.

[4] Keith H. Bennett , Václav T. Rajlich, "Software maintenance and evolution: a roadmap," 2000 Proceedings of the Conference on The Future of Software Engineering, pp.73-87.

[5] Cardoso Jorge, Sheth Amit. "Semantic e-workflow composition," Journal of Intelligent Information Systems,2003,21(3):191～225

[6] S. R. Ponnekanti and A. Fox. "SWORD: A developer toolkit for Web service composition," In Proceedings of the 11th World Wide Web Conference, 2002, pp. 83～107

[7] KEITH M, "From UML to BPEL:Model Driven Architecture in a Web services world," http://www.ibm.com/developerworks/webservices/library/ws-uml2bpel/.

[8] VAN DER AALST W M P., LASSEN K B, "Translating Unstructured Workflow Processes to Readable BPEL: Theory and Implementation," Information and Software Technology, vol. 50, pp.131-159, Feb. 2008.

[9] Sneed, H.M. AneCon GmbH, Vienna, Austria, "Integrating legacy software into a service oriented architecture," Proceedings of the 10th European Conference on Software Maintenance and Reengineering, pp. 11 -14.

[10] Gregor Hohpe, Bobby Woolf, "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, "Addison-Wesley Professional, 2003

[11] Umapathy, K, Purao, S., "Designing Enterprise Solutions with Web Services and Integration Patterns," 2006 IEEE International Conference on Services Computing, pp. 111 – 118

[12] Kajko-Mattsson, Mira; Lewis, Grace A.; Smith, Dennis B, "A Framework for Roles for Development, Evolution and Maintenance of SOA-Based Systems," Proceedings of the 29th International Conference on Software Engineering Workshops, 2007. pp:117

VIII. Biographies

Songlin Hu was born in Shanxi province, China, on Oct. 20, 1973. Songlin received Bachelor and Master Degrees from the Taiyuan University of Technology, and receive PHD degree from Beihang University in 2001..

He is now working in the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His special fields of interest included Service computing, distributed event based system, and Enterprise Application Integration..

Ying Liang was born in Beijing, P.R.China, on May 31, 1962. She graduated with a Bachelor's degree of Engineering from Tsinghua University. She is a senior engineer of the Institute of Computing Technology of Chinese Academy of Sciences. Her current research interests include software integration and service computing.

Jiuming Tian was born in Tengzhou, Shandong Province, P.R.China, on April 8, 1983. He graduated from University of Science & Technology of China in 2005 and now is receiving postgraduate education in the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include workflow, web services composition and BPEL.

Yicheng Song was born in Wuhan, P.R.China, on December 17, 1984. He graduated from Wuhan University in 2006 and now is receiving postgraduate education in the Graduate University of the Chinese Academy of Sciences. His research interests include algorithms design techniques and analysis; Service-Oriented Architecture and the structure of operation system.

Reliability Oriented QoS Driven Composite Service Selection Based on Performance Prediction

Lei Yang Yu Dai Bin Zhang
College of Information Science and Technology, Northeastern University
yanglei@mail.neu.edu.cn

Abstract

In order to improve the reliability of composite service, this paper proposes an approach of reliability oriented composite service selection. Firstly, a service reliability model which integrates request processing reliability and transmission reliability is proposed. In order to compute the transmission reliability, a semi-Markov model based performance prediction is used. Finally, we put the proposed reliability model into QoS driven service selection. Experimentations show that the proposed selection approach have better performance in preserving the reliability of composite service than traditional QoS driven selection approach.

1. Introduction

As web services operate autonomously within a highly variable environment (the Web), as a result of which their QoS may evolve relatively frequently, either because of internal changes or because of changes in their environment [1], such dynamic properties highlight the problem of QoS driven service selection.

Most previous works divides QoS and reliability into two different research fields and studies them separately. However, in reality, QoS and reliability are closely related and affect each other. Let us take one of QoS attributes, response time, as an example. The response time of composite service is a random variable affected by many factors. First, there are many services available on the Web with different request processing speeds. Thus, the response time can vary depending on which service is used for executing the task in the composite process. Second, some service can fail during its execution. So the response time is also affected by the processing reliability. Third, the communication links can fail during data transmission. Thus, the transmission reliability affects the response time as well. Then, when tasks are assigned to services with better response time and worse reliability, the failures of any service and any communication link will make the entire composite service incomplete. This will cause a re-selection to bind new services, which inversely increase the response time of composite service. Thus, this paper attempts to select composite service in order to maximize the QoS of composite service considering the effect of reliability as well.

In this paper, we propose an approach of reliability oriented composite service selection. Key to our approach is the proposed model of service reliability. Such reliability model integrates request processing reliability and transmission reliability, which can be used for evaluate the reliability of services. Meantime, a semi-Markov model is used to quantify the reliability of service. Then, reliability oriented QoS driven composite service selection is presented. Experimentations show that the proposed selection approach have better performance in preserving the reliability of composite service than traditional QoS driven selection approach.

2. Related Works

As web service is a kind of software on the Web, traditional reliability model for evaluating the software, such as Ref.[2], can be used to quantify the request processing reliability. However, as web service existing on the Web, the state of Network (such as load and throughput) will affect the reliability of service. Thus, when it is to model the reliability of service, traditional model for quantifying reliability of software cannot be directly used in the field of service composition.

In the researching field of service composition, researchers [3] use the successful execution rate of a service with the maximum expected time frame to quantify the reliability of a service. This approach is relatively simple. In Ref. [4], the researcher uses a Possion Distribution to quantify the reliability of grid service with the assumption that the data transmission

speed and failure rate of Network is a constant value. However, such assumption do not always hold true. Such reliability model cannot reflect the reliability of service in many situations.

Compared with the above works, we model the service reliability as an integration of request processing reliability and transmitting reliability.

3. Performance Prediction for Reliability Modeling

3.1 Preliminaries

In this section, we introduce some basic concepts that will be used in the remainder of the paper.

Definition 1. QoS of Atomic Service. For an atomic service s (which only contains one operation), the QoS of s can be defined as: $QoS(s)=<Q^t(s), Q^p(s)>$, where:

- $Q^t(s)$ is the response time of s. $Q^t(s)=t_p+R/V_{transmission}$, where t_p is the request processing time; R is the amount of data needed to transmit between s and the execution engine and $V_{transmission}$ is the transmission speed.
- $Q^p(s)$ is the cost of invoking s.

From the definition, the change of request processing time, the transmission speed and the cost will influence the QoS of the service. The affected QoS of service will in turn influence the QoS of composite service. Cost for invoking a service is published by the service provider. The request processing time relies on the number of requests in the waiting list and the processing speed of the computer. The transmission speed is affected by the network. Compared with the changes caused by service providers, changes of the request processing time and transmission speed will be changed more frequently. Such change will affect the performance of service. The reliability of the service is to evaluate the degree to which the service can serve the request as announced QoS.

Thus, in this paper, we will based on the following assumption to model the service reliability:

(a) failure at different service and communication links is independent; (b) from the time of sending request to the service to the time of receiving the result from the service, data transmission speed is a constant value; (c) the failure rate of processing the request is a constant value; (d) the cost for invoking a service is never changed.

3.2 Model of Service Reliability in Context of Composition

QoS reflects how a service can implement the function with certain performance. In context of composition, service reliability represents the degree to which service can serve the request as estimated QoS. In the following, we will discuss how to model the service reliability.

In the context of composition, the earliest start time et_{ij} of component service j for task i in the composite process can be computed as Eq.(1).

$$et_{ij}=\begin{cases}\max\limits_{k<i}\{et_{ku}+t_{ku}\}, k\neq 0\\ 0, k=0\end{cases} \quad (1)$$

Where, task k ($k<i$) is the one that will be invoked before task i and t_{ku} is the response time of candidate service u for task k.

As web service existed in the complex Internet environment, the reliability of service should not only consider the processing reliability as most software, but also take into account of transmission reliability. In this paper, we model the service reliability as an integration of request processing reliability and transmitting reliability. To quantify request processing reliability, common software's reliability model [2] is used. To quantify transmitting reliability, as it relies on the current state of the Network, the holding time in current state and the predicted duration, we quantify the transmitting reliability based on a semi-Markov based performance prediction. In the following, we will give the definition of reliability of service.

Definition 2. Reliability of Service. Reliability of atomic service s in composite service CS is the probability that a request is correctly responded with the estimated time frame, which can be defined as follows: $R(s, CS)=<R_p(s, CS), R_t(s, CS)>$, where

- $R_p(s, CS)$ is the request processing reliability, which can be defined as Eq.(2):

$$R_p(s,CS)=e^{-\lambda t} \quad (2)$$

Where, λ is the failure rate of service when processing a request; t is sum of response time of s and earliest start time of s in CS; Eq.(2) is a common in software's reliability, which has been justified in both theory and practice [2].

- $R_t(s, CS)$ is the transmitting reliability. We will show how to quantify it in section 3.3.
- The overall reliability of service s can be computed as Eq.(3)

$$R(s,CS)=R_p(s,CS)*R_t(s,CS) \quad (3)$$

Definition 3. Reliability of Composite Service. Reliability of composite service CS is the probability of that all the component service execute correctly, which can be defined as Eq.(4):

$$R(CS)=1-\prod_v(1-R(s_v,CS)) \quad (4)$$

Where, s_v is a component service of CS.

3.3 Performance Prediction for Quantifying Transmission Reliability

We introduce discrete time semi-Markov model [5] for the prediction.

Definition 4. States of Data Transmission Speed. We use th_V_Q to signify the threshold of data transmission speeds in Qualified state.

- If $V(t)>= th\_V_Q$, then $ST(t)$ =Qualified state;
- If $0<V(t)<=th\_V_Q$, then $ST(t)$=Soft Damage state;
- If $V(t)=0$, then $ST(t)$=Hard Damage state.

Definition 5. Semi-Markov Model for Data Transmission Speed. Let Ω be the state space of data transmission speed Ω={1, 2, 3}. $Z=\{Z_t;\ t>=0\}$ is the random procedure on Ω. If the following conditions are true, we call that $Z=\{Z_t;\ t>=0\}$ is a semi-Markov process.

- If current state is i, the next state will be entered is j with probability P_{ij}. Especially, $P_{ii}=0$;
- Given that the next state entered will be j, the time it spends at state i until the transition occurs is a holding time t with distribution $F_{ij}(t)$.

Let $H_i(t)$ be the distribution of holding time in state i, $H_i(t)=\sum_j F_{ij}(t) * P_{ij}$. The average holding time in state i can be signified as μ_i. According to lemmas [5] of semi-Markov model, there exists stationary distribution $\pi=[\pi_1, \pi_2, \pi_3]$ and for each π_j, it can be computed as Eq.(5). Also, let P_i the steady-state occupancy probability of state i, it can be computed as Eq. (6).

$$\pi_j = \sum_{i=1}^{3} \pi_i P_{ij}; \sum_{i=1}^{3} \pi_i = 1 \quad (5)$$

$$P_i = \frac{\pi_i \mu_i}{\sum_j \pi_j \mu_j} \quad (6)$$

In order to predict the future state, it is required to get the context related to data transmission speed.

Definition 6. QoS-Related Context. The QoS-related context observed by observation o can be defined as $QC(o)=<t_{ob}, v, st_{ob}>$, where t_{ob} is observing time; v is the observed data transmission speed at t_{ob}; st_{ob} is state.

The aim of prediction can be described as: if the current state is i, current time is t and the holding time in current state is d, we need to predict the probability of the data transmission speed V_f at future time t_f above the expected speed V_e. Let j be the state V_e belongs to. To solve this problem, we will consider the following two situations:

- State j is same to i

In this situation, the probability can be a sum of the probabilities in the situations with no transition from t to t_f and situation with at least one transition. Then, the probability can be computed as Eq. (7).

$$\begin{aligned} &P((V>V_e)\wedge(D_i>d)) \\ &=P((V>V_e)\wedge(D_i>t_f-t+d|D_i>d)) \\ &+P((V>V_e)\wedge(Z_{t_f}=i)\wedge(d<D_i<d+t_f-t|D_i>d)) \\ &=(1-F_i(V_e))*\frac{1-H_i(t_f-t+d)}{1-H_i(d)}+(1-F_i(V_e))*P_i*\frac{H_i(t_f-t+d)-H_i(d)}{1-H_i(d)} \end{aligned} \quad (7)$$

- State j is different from i.

If state j is different from i, it means that there exist at least one transition during the duration from t to t_f. Then, the probability can be computed as Eq. (8).

$$\begin{aligned} &P((V>V_e)\wedge(Z_{t_f}=j)\wedge(d<D_i<d+t_f-t|D_i>d)) \\ &=P(V>V_e)*P(Z_{t_f}=j)*P(d<D_i<d+t_f-t|D_i>d) \\ &=(1-F(V_e))*P_j*\frac{H_i(t_f-t+d)-H_i(d)}{1-H_i(d)} \end{aligned} \quad (8)$$

According the prediction of transmission speed, the transmission reliability of service s in the context of composition, can be computed as Eq.(9):

$$R_t(s,CS)=P \quad (9)$$

Where, P can be either computed as (7), or (8), according to the current state of service.

4 Reliability Oriented QoS Driven Composite Service Selection

Reliability oriented QoS driven composite service selection is to maximize the QoS of composite service considering the effect of reliability as well. The problem of such selection can be described as (10).

$$\begin{aligned} &\max_v \ F(CS_v) \\ &s.t\ \ Q^t(CS_v)\le Q_c^t \\ &R(CS_v)\ge R_c \end{aligned} \quad (10)$$

Where, $CS_v=\{(t_1, s_{1,k1}),\ldots, (t_i, s_{i,ki}), \ldots, (t_n, s_{n,kn})\}$, here, t_v is the task in composite process and $s_{v,kv}$ is the service selected for tasl t_v; Q_c^t is the constraint of response time; R_c is the reliability constraint; F is the fitness function which considers both QoS and reliability of the composite service and can be computed as (11).

$$F(CS_v)=w_t*\left(\frac{Q^t(CS_v)-u_t}{\sigma_t}\right)+w_p*\left(\frac{Q^p(CS_v)-u_p}{\sigma_p}\right)+w_R*R(CS_v) \quad (11)$$

Where w_t, w_p and w_R are the weights ($0<=w_t, w_R, w_p<=1$, $w_t+w_p+w_R=1$). σ and μ are the standard deviation and average of the QoS values for all potential composite services.

Such problem can be mapped into a multi-constraints satisfying problem. Several approaches [6] can be used for solving such problem. As the limitation of this paper, we will not discuss how to use the algorithm to solve such a problem in detail.

5. Experimentations

Experimentation 1 is used to test the effectiveness of the proposed semi-Markov model based QoS predicting approach. Simulate test set of data transmission speed according to the Gaussian distribution. The threshold of failure probability is 0.9. The size of QoS-related contexts is 100000. Compare the relation among predicted result, predicting interval and the observation interval between two neighboring contexts. Table.1 gives the result (O_Q is the observation interval between two neighboring QoS-related contexts in *QCS*; *N* is the number of predictions; *R* is the average accurate rate of the predictions.

Table. 1. Semi-Markov Based Predicted Result

	O_Q=0.5s			O_Q=0.1s			O_Q=0.05s		
	I= 10	*I*= 60	*I*= 180	*I*= 10	*I*= 60	*I*= 180	*I*= 10	*I*= 60	*I*= 180
N	300	300	300	200	200	200	150	150	150
R%	95	86	80	97	93	83	98	95	92

Table.1 shows that if the observation interval between two neighboring contexts is smaller and the predicting interval is shorter, the prediction will be more accurate. When the observation interval is short enough, although predicting interval is a little bigger, the accurate of prediction will be better also. Thus, through minimizing observation interval, the accuracy of prediction result can be improved.

Experimentation 2 is used to test the reliability of composite service. Randomly generate 10 abstract composite processes. Select the composite service for each composite process. Simulate test sets of data transmission speed according to the Gaussian distribution and set I_Q=0.1s, the threshold of reliability is 0.95. Simulate change of data transmission speed according to Gaussian distribution from the beginning time of composite service execution to the completing time of composite service execution, after selection. Compare the response time of composite service. The result is shown in Fig 1.

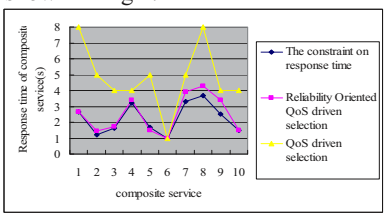


Fig. 1. Comparison of Re-Selection Numbers

Fig 1 shows that the composite service selected by the proposed reliability oriented QoS driven service selection approach always has better response time than the one selected by QoS driven approach. This is because that during the selection process, not only the QoS of composite service is considered, but also the reliability of composite service is respected in our approach. Thus, the more reliable composite service will always meet the constraint of response time.

The above experimentations show that the proposed approach is more effective in preserving the reliability of composite services.

6 Conclusions

In order to maximize the QoS of composite service considering the effect of reliability as well, we propose an approach of reliability oriented composite service selection. Model of service reliability is proposed. Then, we put such model into the problem of reliability oriented QoS driven composite service selection. Experimentations show that the proposed selection approach have better performance in preserving the reliability of composite service than traditional QoS driven selection approach.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under Grant No.60773218.

References

[1] L. Z. Zeng, B. BENATALLAH, "QoS-Aware Middleware for Web Services Composition", *IEEE Transactions on Software Engineering*, 2004, 30(5), pp.311-327.

[2] M. Xie, Y.S. Dai, and K.L. Poh, "Computing Systems Reliability: Models and Analysis", Kluwer Academic, 2004.

[3] G. Huang, L. Zhou, X.Z. Liu, H. Mei and S.C. Cheung, "Performance Aware Service Pool in Dependable Service Oriented Architecture", *Journal of Computer Science and Technology*. 2006, 21(4), pp.565-573.

[4] Y.S. Dai, G. Levitin, and K. S. Trivedi, "Performance and Reliability of Tree-Structured Grid Services Considering Data Dependence and Failure Correlation", *IEEE Transactions on Computers*. 2007, 56(7). pp.925-936.

[5] M. Malhotra, A. Reibman, "Selecting and Implementing Phase Approximations for Semi-Markov Models", *Communication Statistics-Stochastic Models*, 1994, 9(4), pp. 473-506.

[6] R. Fleteher, "Practical Methods of Optimization", New York, John-Wiley and Sons, 1981.

Design of an RSS Crawler with Adaptive Revisit Manager

Bum-Suk Lee[1], Jin Woo Im[1], Byung-Yeon Hwang[1], and Du Zhang[2]

[1] *Department of Computer Scinece and Engineering, The Catholic University of Korea, 43-1 Yeokgok 2-dong, Wonmi-gu, Bucheon-si, Gyeonggi-do 420-743, Republic of Korea*
{bslee,cukfan,byhwang}@catholic.ac.kr
[2] *Deptartment of Computer Science, California State University, Sacramento, U.S.A.*
zhangd@ecs.csus.edu

Abstract*—RSS (Rich Site Summary, or Really Simple Syndication) is widely used for notifying readers of updated information on blogs and feeding news to readers quickly. RSS is very simple, and so is mostly used as a web service. However there is no satisfactory search engine which works for RSS. The reason is that RSS is continuously modified, and the structure of general search engines is ineffective to collect information from RSS sources.*

In this paper, we discuss a web crawling algorithm, and propose a structure for an RSS crawler which is geared toward collecting and updating RSS in the Web2.0 environment. The proposed method (1) uses visited domain name history to predict the location of the RSS of a new seed URL, and (2) updates RSS information adaptively, based on some update-checking heuristics. These approaches can serve as cornerstones for an efficient and effective RSS search engine.

Keywords*—Web2.0, RSS, Crawler, Adaptive Revisit manager*

1. Introduction

Web2.0 [1] is a next generation web service paradigm which is different from past technologies. In this paradigm, information providers and users are engaged in interactive communications about their demands and needs [2]. RSS (Rich Site Summary or Really Simple Syndication), a novel XML based technique in the Web2.0 environment, is used for transferring data easily, and notifying readers of updates to blogs via RSS reader applications or meta-blogs almost in real-time.

Historically, RSS has been used to refer to "Rich Site Summary," "RDF Site Summary" and "Really Simple Syndication." It is a data format based on XML, used to supply update information on websites with frequent content updates [3]. RSS is very simple, and so is mostly used as a web service especially on blogs. People can subscribe to an RSS "feed" with an RSS reader application. The feed lets people know that new information is available even though they do not visit the website, and people can use the reader application to categorize contents to their taste.

In response to the increasing use of RSS, sites have emerged which offer an RSS reader-like portal service, such as "allblog.net" and "naaroo.com" in Korea. These sites are called meta-blogs. People can add their RSS address to the meta-blogs by themselves. The meta-blogs check updates of registered blogs and categorize contents like news articles for portal sites. However the meta-blog is inconvenient because they involve passive registration of RSS by personal bloggers. Also people can only search for registered contents, and so the search results of portal contents are limited to narrow topics.

Development of a search engine specifically for RSS is needed to solve these problems. Generally speaking, a web search engine consists of three parts: a crawler, an indexer, and a searcher [4,5]. The crawler collects information from websites, and the indexer manages an index of gathered information to support fast retrieving. The searcher offers searching results based on the index. There are studies on RSS gathering, but the results have been less than satisfactory thus far. In this paper, we propose an RSS crawler that takes advantage of the features of RSS. This proposed RSS crawler makes use of gathered domain information to find the location of RSS from seed URLs. Furthermore it adaptively checks the updates from sites of collected RSS with an RSS manager. To do this, the RSS manager keeps information on the update time of each RSS feed.

The rest of the paper is organized as follows: Section 2 offers a brief overview and related work of the search engine and RSS. Section 3 describes the proposed RSS crawler with emphasis on the RSS crawling algorithm for effectively gathering RSS update information and on the structure of the RSS manager for adaptive updates of RSS feeds. Section 4 gives a comparison between our proposed approach and some existing work. Finally, in Section 5, we conclude the paper with remarks on future works.

2. Related Work

2.1. Web Crawler and RSS Search Engine

One of the most important reasons for Google to become a tremendously successful company in the web

service field is that Google has developed a search engine that crawls the enormous web very effectively. Google uses a totally automatic crawler and a specialized page-rank algorithm [6]. The days to register and categorize website information manually by humans are long gone. Most of the web search engines nowadays use some sort of automatic crawling algorithms.

There are researches for developing RSS crawler by various researching group [7-10]. Among them, beta version of Feedshow.com was developed by French researchers in 2006. It gathers RSS with crawler and provides a search engine. Furthermore Bloglines.com and Blogpulse.com are developed with their own RSS aggregator. However most of these services cannot adequately satisfy people's demands and needs, because the searching results are too limiting to fine useful information. Also ranges of the results are limited to their locality, such as Europe or the States. It seems that the problem stems from the lack of some effective crawling algorithm for aggregating RSS and from the disconnection between the web service and the features of RSS which update frequently.

There is also interest in developing the RSS indexer. Brooks and Montanez introduced autotagging and hierarchical clustering into the blogosphere [11]. The automatically generated tags were useful to help users with their choices. However these tags could not be combined with each other in any meaningful relationships, so people had to connect them by hands. It is acknowledged in the paper that the autotagging algorithm needs to be improved.

2.2. Features of RSS

RSS is updated when new content is added on a website or blog. The period of the updates is determined by the preferences of the information provider or the character of the service type. RSS can be located in the same URL-subpath as the blog in some cases, such as WordPress (an installable type blog) and Blogger (a joinable type blog), which are the two most popular blogging tools in the world.

The period of update is important in the collection of information from RSS. The internet has been growing consistently over the past two decades, and a large number of web pages are newly created every day. Furthermore the bloom of blogs results in new contents every day, with frequent updates all over the world. These voluminous newly created web pages far exceed the ability of search engines to crawl and process, so the revisit period for a particular website to check its update may be far longer than the update period of the site itself. This causes deterioration in the quality of search results, because the results may not contain the most recently updated contents until the crawler revisits the websites that appear in the search results.

This situation gets worsened especially in blogs. Many people post their thoughts about life and social issues, and often discuss these issues with other people. However many events happen everyday, so information and issues that people need and are interested in change fast.

Precision and recall are the popular evaluation measures for search engines. Precision is the percentage of related documents with keywords in the results. Recall is the ratio of number of related documents with matching keywords returned by a search to the whole number of related documents. A search engine with a long revisit period often cannot show newly updated information. As a result, the recall quality of the search engine deteriorates. On the other hand, while frequent revisits can improve recall quality, they cause unnecessary network traffic and maintenance costs.

A main feature of RSS in both installable type of blogs and joinable type of blogs is that RSS is located in the same subpath as the blog, and so we can gather RSS easily based on the domain information without visiting all URL links. With the addition of a new module to use domain information, we can improve crawling speed.

3. Proposed RSS Crawler

3.1. Gathering RSS with Domain Information

The main use of RSS is for blogs and news feeds. Between these two types of web services, blogs use RSS mostly widely. Accordingly, we can design an RSS crawler which gathers RSS by considering the typical features of blogs. The most commonly used blogs belong to either the installable and joinable type, such as WordPress or Blogger. In installable type blogs, people install the blog application on a hosted site in their own domain, while people join and get sub level domain access in the joinable type.

People have their own domain and web server in the case of installable type blogs. For example, WordPress has to be installed on a web server, and so its RSS is located in the same sub-path, assuming the owner did not change when they installed it. This feature is also applied with other kinds of installable type blogs. Therefore when the crawler visits a blog, it can recognize which kind of installable blog application is used while reading any web pages on that domain. With proper storage and indexing of these kinds of information, the crawler can find the location of an RSS without visiting all links.

Blogger is a representative example of the joinable type of blogs. In this case, the blog creates a sub level domain with a user ID, so if a user joins with ID “tom” then, the address of the blog might be generated as “http://tom.blogspot.com/”. Joinable blogs which generate sub level domains have a feature that allows RSS to be found in the same sub path location as the blog.

In this paper, we propose a path manager that uses the domain and path information when it crawls. The path manager analyzes and stores information about current

pages, and predicts the location of RSS in the current domain. The structure of the proposed crawler is shown in Figure 1. The crawler in Figure 1 has two features which are different from normal search engine crawlers: (1) a part for comparing domain information, and (2) a part for checking whether the blog is an installable type or a joinable type.

In the first step, the path manager checks that there is a visited upper level domain based on the top-level domain and second-level domain. This helps the crawler find RSS quickly in the case of joinable type blog. Next, the crawler checks whether the blog is an installable type, if there is no matched domain. The crawler identifies an installable type blog based on the pattern of path links and text. These can be analyzed and maintained during the crawling process. If the seed URL is based on either an installable or joinable type of blog, the crawler may find the location of RSS, and it can reduce unnecessary visits to find RSS.

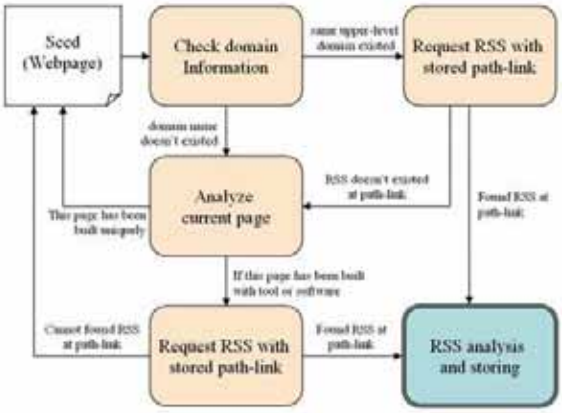


Figure 1. A structure of crawler for gathering RSS.

3.2. Adaptive Revisit Manager

In this subsection we describe an adaptive revisit method to check updates of RSS based on the update period of gathered RSS. Generally speaking, web crawlers revisit gathered websites occasionally to check updates and find errors in URL connections. RSS is more efficient on websites which are frequently updated than less visited websites. Furthermore information consumers are greatly satisfied when the RSS always includes recent information. Accordingly, the meta-blog and RSS reader applications have to provide recent information to satisfy people. The proposed crawler analyzes the <pubDate> element in gathered RSS, and uses it to predict the next update time. The <pubDate> element includes a date and time in the form: "<pubDate>Wed, 16 Jan 2008 01:51:44 +0900</pubDate>". With this information we can determine when the website is updated: daily or on specific day such as on Saturday. The crawler predicts next update time based on analyzed data, and revisits the RSS at that time. This adaptive revisit method is based on the posting pattern of the information provider.

The proposed crawler considers three pieces of information in the <pubDate> element: (1) The update time interval. The crawler finds the time interval of update, e.g., daily or weekly. This can be calculated from date information. (2) The day of the week. Posting a new article is the result of human work. People have their own life patterns, so the analysis of the day of the week can be an important clue. (3) The update time. A frequently updated time is also based on human patterns, and so we can expect that it is an effective indicator. Hence the crawler can revisit during frequently updated time to reduce the gap between posting time and revisit time. Based on the aforementioned heuristics we can expect an improvement in the recall quality of our search results.

Next we introduce an analysis method to be utilized by the adaptive revisit manager. This method analyzes gathered RSS files and applies a statistical method on the day of week and an adaptive method on time. Let P_u denote the probability of future day's update. Let U and $\bar{U}$ be the value of recent updated data (U is constantly 1) and the mean of past updated data respectively. The algorithm estimates the $\bar{U}$ using $\bar{U}=\delta_u/\delta_w$ where δ_w is a whole period for all data and δ_u is a number of days where there exist updated data. Table 1-(a) shows the presence or absence of updated data in a sample RSS. In this example, δ_w is 45 and δ_u is 25, so $\bar{U}$ is almost 0.556.

Table 1. Presence or absence of update in a sample RSS.

Sat.	*Sun.*	*Mon.*	*Tue.*	*Wed.*	*Thu.*	*Fri.*
1	*0*	*0*	*0*	*1*	*1*	*0*
1	*0*	*0*	*0*	*1*	*0*	*1*
1	*1*	*0*	*1*	*0*	*1*	*1*
1	*1*	*1*	*1*	*1*	*0*	*0*
1	*1*	*1*	*0*	*1*	*0*	*0*
0	*1*	*0*	*0*	*1*	*0*	*0*
1	*1*	*1*				

(a)

Sat.	*Sun.*	*Mon.*	*Tue.*	*Wed.*	*Thu.*	*Fri.*
6/7	*5/7*	*3/7*	*2/6*	*5/6*	*2/6*	*2/6*

(b)

The term α is a weight factor that controls the rate of convergence of the algorithm (always at 0.9), and β is a statistical weight factor that is calculated from the mean of update on the day of week. The β can be calculated according to Table 1-(b). The statistical weight factor of Tuesday which is a next day of the latest update is 0.333(2/6) in this example. Finally the probability is estimated using $P_u=\alpha\beta U+(1-\alpha)\bar{U}$. We determine 0.5 as the threshold. Here, the probability value is 0.355. It is smaller than the threshold, so we can predict that there may not exist an update on the next day. After that the algorithm recalculates an update probability of the day after tomorrow (Wednesday). In this case δ_w increases to 46. With the same method, $\bar{U}=25/46 \approx 0.543$ and β is 0.833(5/6), and so P_u is

0.804. We know that most likely there will be an update on Wednesday.

4. Comparison

In this section we compare our approach with some of the similar web services. Table 2 summarizes the comparative results. The similar web service providers include: Google, Feedshow, Bloglines, Blogpulse, and Allblog.

Table 2. A comparison with similar web services.

	Search Engine with Crawler					Meta-blog
	Our approach	Google	Feedshow	Bloglines	Blogpulse	Allblog
Method for collecting RSS	**RSS crawler**	web crawler	RSS crawler	RSS crawler	RSS crawler	register RSS by hands
Method for checking update	**adaptive**	static	static	static	static	adaptive
Times until Update reflection	**short**	long	long	short	normal	normal

As we mentioned in this paper, our approach consists of two major components. (1) The crawler that crawls RSS efficiently with gathered domain information. It reduces useless out-links so as to find RSS fast. (2) Revisit manager that is based on an adaptive and statistical method for checking updates. This helps to reduce the cost for checking the updates, and to promptly reflect the updates as soon as they occur. Google has absolutely nice performance but its method for checking updates takes too much time. Performance of Feedshow is very poor while Bloglines and blogpulse are in good performance range. However Bloglines checks updates once an hour and Blogpulse checks it once a day, thus creating unnecessary checking cost. Allblog is a different kind of service. It is a meta-blog, and people register their RSS to share with others. Its checking update strategy is adaptive according to frequency of RSS update. On balance, our approach can provide efficient performance with low update-checking cost.

5. Conclusion

RSS is the most widely used information distribution technique in the Web2.0 environment. It is an open standard for transferring data efficiently based on XML. RSS is widely used in many fields such as news and blog feeds, and data transferring on mash-up services. However, the state-of-the-practice in the field is that few existing RSS search engines is efficient and effective. As a result, development of better RSS search engines is critically needed.

In this paper, we described the design of an RSS crawler which gathers and updates RSS efficiently. The crawler, the main part of a search engine, is a tool to visit the vast World Wide Web and collect data. Commonly the crawler recursively analyzes out-links of web pages starting from a seed URL, but this method wastes too much time and network traffic to work efficiently for RSS. Furthermore we wanted to design a crawler just for gathering RSS effectively. The proposed RSS crawler in this paper has two main features: (1) It gathers RSS easily based on path and domain analysis of installable and joinable type blogs, which represent the majority of RSS use. (2) It revisits gathered RSS adaptively based on update pattern analysis of the period, day of the week, and time of updates on the target site.

The proposed RSS crawler appears to gather RSS more efficiently, and the adaptive revisit method can improve recall quality of the search results. The proposed method offers the possibility of reducing wasted time on crawling, and improving recall quality. For future work, we intend to implement the proposed RSS crawler and evaluate its performance.

References

[1] T. O'Reilly, "What Is Web2.0: Design Patterns and Business Models for the Next Generation of Software," self published on www.oreilly.com, 09/30/2005.

[2] D. E. Millard and M. Ross, "Web2.0: Hypertext by Any Other Name?," In Proc. of ACM Conf. on Hypertext and Hypermedia 2006, pp. 22-25, 2006.

[3] RSS version 2.0 Specifications, http://blogs.law.harvard.edu/tech/rss, 2003.

[4] S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," Computer Networks, Vol. 30, No. 1-7, pp. 107-117, 1998.

[5] N. Blaž, "A Survey of Focused Web Crawling Algorithms," In Proc. of the Conf. on Data Mining and Warehouses, 2004.

[6] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," Unpublished Manuscript, 1998.

[7] I. Rose, R. Murty, P. Pietzuch, J. Ledlie, M. Roussopoulos, and M. Welsh, "Cobra: Content based Filtering and Aggregation of Blogs and RSS Feeds," In Proc. of the 4th Symposium on Networked Systems Design and Implementation, pp. 29–42, 2007.

[8] X. Li, J. Yan, Z. Deng, L. Ji, W. Fan, B. Zhang and Z. Chen, "A novel clustering-based RSS aggregator," In Proc. of 16th Int'l Conf. on WWW, pp. 1309-1310, 2007.

[9] S. Buraga and T. Rusu, "Search Semi-Structured Data on Web," In Proc. of the 7th Int'l Symposium on Automatic Control and Computer Science, 2001.

[10] D. Chmielewski and G. Hu, "A Distributed Platform for Archiving and Retrieving RSS Feeds," In Proc. of the 4th Annual Int'l Conf. on Computer and Information Science, pp. 215-220, 2005.

[11] C.H. Brooks and N. Montanez, "Improved Annotation of the Blogosphere via Autotagging and Hierarchical Clustering," In Proc. of the WWW2006, 2006.

QuickPay Online Payment Protocol

Jian Dai Mark Stamp
Department of Computer Science
San Jose State University

ABSTRACT In this article, we propose a new online payment protocol, QuickPay, which is built as a middleware framework for online payment and other online financial transactions. We provide the general design of the QuickPay system by describing its business and technical goals, system architecture, major players, and the relationships among the players. To show how QuickPay works, we discuss one specific application in some detail. We demonstrate that QuickPay can provide an effective and secure for online payments.

Keywords
Security, online payment, micropayment

INTRODUCTION

Online payment systems are designed to facilitate online purchases involving various amounts of money [3, 5, 8]. Online payments present several challenges, such as Trust between physically separated customer and vendor; anonymity of customer; instant payment; low processing cost; etc. [6, 9].

At present, many online payment systems have been developed. Based on their design principles and characteristics, we can roughly classify them into the following three categories: Token-based Systems [2 8, 7, 10, 12]; Account-based System [1]; and Protocol-based Systems [4].

Based on our analysis of existing online payment systems, we propose QuickPay as a generic online payment application which, in our estimation, provides an optimal tradeoff among the competing demands for security, effectiveness and cost. QuickPay has also been designed so that it can be used for the widest possible variety of commercial activities.

QUICKPAY REQUIREMENTS

We believe the following business requirements are the most critical to the success of a system such as QuickPay.

Openness - QuickPay can accommodate users who use other online payment systems.
High scalability - QuickPay is able to support additional customers and vendors without any impact on existing customers and vendors. This implies that there is no performance bottleneck in the system.
Independence and open plugin - Here, we mean that QuickPay supports different online financial transaction with different sets of protocols. Each set of protocol can plugin to the framework freely and independently.
Isolation - By isolation, we mean that QuickPay's users (vendor, broker, or customer, as discussed below), can have their own implementation for QuickPay's protocols. Their implementations are isolated from each other.
Security at low cost - This implies that QuickPay provides reasonable security (as compared to existing online payment systems), with a low overall cost.
Standard user interface - This means that QuickPay will have a standard user-friend interface for end users.

ARCHITECTURE

QuickPay is designed as a middleware application based on HTTP, Web Services, and other related open source protocols. In addition, we assume that in most cases, customers do online shopping from their own dedicated domains, such as home computers so that we can tie customer "identification" with specific machines. Secondly, we assume that a broker (who is responsible for settling accounts) has much less motivation to cheat as compared to vendors and customers, due to the broker's long-term interest in commissions.

QuickPay defines four main player roles for the system, as discussed below and illustrated in Figure 1.
System Host is the central controller and administrator for the whole QuickPay system. This is the only component that can be trusted by

all other players and it has the most authentication information on the other parts.
Vendor represents anybody (usually an ecommerce website) that is selling products (tangible or intangible) online and intending to use QuickPay to collect payments.
Financial Broker is an entity (typically, a financial institution) that serves to facilitate payments for online purchases on behalf of its customers (i.e., End Users in QuickPay). A Financial Broker is a trusted partner and is expected to have a long-term relationship with the System Host.
End User.is the final player role in QuickPay. In business term, an End User is generally regarded as a person who wants to use QuickPay to pay for his or her online purchase. In technical term, an End User represents an installation of QuickPay's client-side software, which is bonded to a machine (desktop, laptop, Palm, Internet Phone, or other).

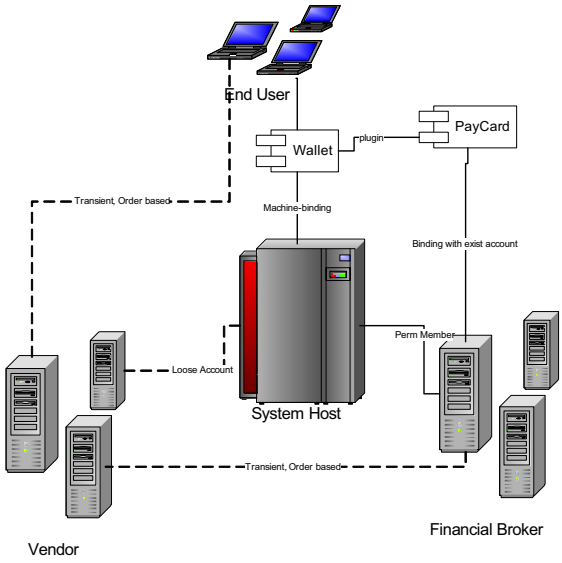


Figure 1

In addition to these player roles, the QuickPay system also defines two client-side software components. The **Wallet** is a browser plug-in software component provided by the System Host. It provides a standard QuickPay client side graphical user interface; maintains the necessary client-side data; and handles all of the online communication between End Users and other parts of the system. A **PayCard** is a Wallet plug-in software component provided by the Financial Broker.

Among these players, the System Host and a Financial Broker have the closest and most fully trusted relationship. Instead of serving End Users directly, QuickPay is designed to work with a Financial Broker, which in turn serves its End Users.

The System Host knows an End User based on the End User's Wallet, which contains a unique identifier assigned by the System Host and is bonded to a single physical machine. The Financial Broker deals with its End Users via PayCards. An End User obtains a PayCard if and only if he or she has an existing account with the host Financial Broker.

Anyone who wants to sell a product or service online can join QuickPay as a Vendor and use QuickPay to collect payments. Theoretically, all a Vendor needs is a unique Vendor ID from the System Host. Note that the Vendor only has an order-based one-time relationship with End Users and Financial Brokers.

QUICKPAY PROTOCOLS

As mentioned above, QuickPay is composed of a set of independent protocols. Considering the fact that it is not possible to cover all the protocols in this short article, we will briefly introduce the support protocols and then use the micropayment protocol as an example to describe some of the most common characteristics of the protocols. See [13] for complete details of the QuickPay protocol.

QuickPay Support Protocols QuickPay defines a set of support protocols. Their main purpose is to take most of the security burden from application protocols so that those application protocols can perform with relatively high security at low cost. For example, in QuickPay, an End User is designed to pay for an order to its Financial Broker indirectly. To support this feature, QuickPay employs an administration protocol, Add-PayCard, which serves to bond an End User (represented by its Wallet) to an existing account in an individual Financial Broker (represented by its PayCard).

Generally, administration protocols run much less frequently but are more critical for overall system security as compared to application protocols. Consequently, the implementation of these protocols can rely on high-security algorithms with relative high cost.

Example: Micropayment protocol. The Micropayment protocol must run at very low transaction cost in order to support mini-sized online orders. To achieve this goal, the QucikPay micropayment protocol is designed with the emphasis on performance over security. In addition, QuickPay shifts some security burden to administration protocols in order to improve the performance of this protocol.

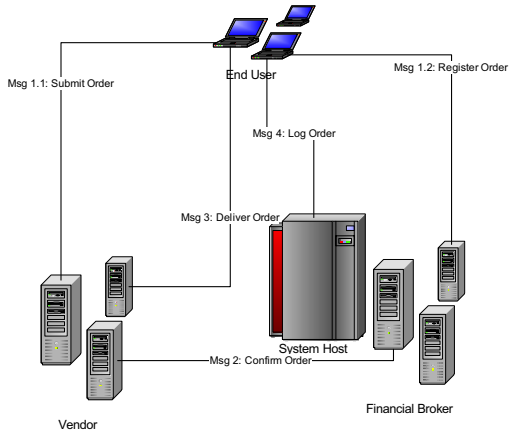


Figure 2

The QuickPay micropayment protocol consists of four steps and 5 messages as illustrated in Figure 2. The process starts when the End User submits an order to a Vendor, as represented by Message 1.1 in Figure 2. Note that this message contains information about the active Financial Broker. Simultaneously, the End User registers the order with the Financial Broker with Message 1.2, the integrity of which should be guaranteed by Financial Broker. After receiving the message, the Vendor will send a confirmation message (Message 2) to the Financial Broker. If the Financial Broker finds that the order has been registered by the End User, it commits to pay for this order on behalf of the End User; otherwise, it will deny the order. The Vendor will then deliver the order to the End User (Message 3) only if it obtains a commitment for payment from the Financial Broker; otherwise, it will deny the order. Once the order has been delivered to the End User, the End User will send a log message (Message 4) to the System Host so that System Host can track the order.

In this Micropayment protocol, all of the online transactions are public and in plaintext format. There is no direct security implementation applied to the transaction of these messages. The payment commitments of the Financial Broker made here are temporal, and will be finally settled in a batch settlement process. Note that the System Host is not directly involved in the payment transaction, but it does monitor the whole process indirectly by keeping track of orders.

DISCUSSION

As mentioned above, the central issue faced by online payment systems is how to achieve a reasonable tradeoff between transaction costs and payment security. Instead of focusing on trying to protect sensitive data at low costs with specialized algorithms, QuickPay is aimed at cutting the transaction cost (and, simultaneously, the value to the attacker) of the sensitive data that is transmitted. To accomplish this, QuickPay implements a set of unique strategies.

Unlike account-based payment system, QuickPay uses device-base (such as IP address) authentication. This greatly reduces the overhead of a transaction. As well known, spoof attack is the main security of such authentication. Besides taking advantage from built-in protection in standard protocol (such as TCP) and the fact that it is difficult for spook attackers to get response for their false request, QuickPay also designed additional protection for such attack. For example, the order-registration design can be used by Financial Broker to filter DoS attack and deny forgery orders from impersonate Vendor. The offline support of System Host can be used to reduce DoS attack against it.

QuickPay supports a finite set of Financial Brokers, whose information is public and static. Therefore, it is easy for a Vendor to identify a Financial Broker and it is difficult for an attacker to commit a false payment by impersonating a Financial Broker.

QuickPay requires a Financial Broker to guarantee the integrity of the transaction between its Payment Card and its host, and the Financial Broker only commits to a registered order. An attacker cannot make a forgery of the order unless he or she can register it, which would, in practice, be very difficult, if not impossible.

For most existing payment systems, the transaction process is linear. If any step in the payment chain is broken, the system is compromised. Suppose that such an application has two steps. If the security risk for each step is

5%, then the overall risk would be almost 10% since

$$10\% \approx 9.75\% = 1-(1-0.05)(1-0.05)$$

In contrast, QuickPay uses a collateral verification strategy. That is, for each payment request, the Financial Broker obtains information from both the End User and the Vendor. Under this approach, to obtain a 10% security risk for the overall process, each step in QuickPay can afford a security risk of about 33% to each component, since

$$10.9\% = 0.33*0.33$$

In other words, QuickPay can tolerate a much looser security implementation, which allows for much lower transaction costs, as compared to traditional payment systems.

.

Protection of passwords and other crucial user information is one of the most critical security issues (especially in account-based system), and it is costly to protect users' personal information during a payment transaction. In QuickPay, all of the players except for the Financial Broker do not have End User personal and/or Financial Information and there is no need to exchange such information during payment transactions. As a result, QuickPay itself bears no direct cost to protect such security-related information.

With many existing payment system, customers need to submit some credentials to vendors and those credentials potentially can be abused by malicious vendors to, for example, make fraudulent transaction. In QuickPay, the payment commitment is based solely on individual orders. Consequently, a commitment cannot be readily reused to collect any additional fraudulent payment.

CONCLUSION

In this article, we outlined QuickPay, an efficient online payment protocol designed for different types of online payment and other financial transactions. By identifying and focusing on the most common characteristics of different online payment situations, QuickPay can significantly outperform existing online payment applications. We believe that QuickPay satisfies the core requirements necessary to become a widespread financial solution for a variety of online business activities.

REFERENCES

[1] Anonymous, Case Study: PayPal, http://digitalenterprise.org/cases/paypal.html, September 30, 2004.

[2] Chi, Ellis "Evaluation of micropayment Schemes" HP Laboratories Technical Report, 97-14, pp 1-29, Jan, 1997.

[3] Hallam-Bakeer, Phillip, "Micro Payment Transfer Protocol", http://www.w3.org/TR/WD-mptp-951122, 1995.

[4] IBM, "IBM Multi-Payment Framework (version 1.2)" http://www-306.ibm.com/software/genservers/commerce/payment/mpf.pdf, 1999.

[5] Manasse, M. "The Millicent protocols for electronic commerce" proceedings of the 1st USENIX workshop on Electronic commerce, 1995.

[6] MacKie-Mason, Jeffrey K. and White, Kimberly, "Evaluating and Selecting Digital Payment Mechanisms," in Interconnection and the Internet, G. Rosston and D. Waterman, eds. Lawrence Erlbaum, 1997: 113-134.

[7] Micali, Silvio. and Rivest, Ronald L. "micropayment Revisited" CT-RSA 2002.

[8] Palmer, Jonathan W. and Eriksen, Lars Bo "Digital newspapers explore marketing on the Internet" Communications of the ACM, Volume 42 No 9 pp 33-40, September 1999.

[9] Pilioura, Thomi "Electronic Payment Systems on Open Computer Networks: A Survey" work paper http://wwwi.wu-wien.ac.at/public/ehandel/Zahlung_Ueberblick.pdf 1999.

[10] Rivest, Ronald L. and Shamir, Adi "PayWord and MicroMint: Two simple micropayment systems" Lecture Notes Computer Science, 1318, pp 307-314, 1998.

[11] Shirkey, Clay, "Fame vs Fortune: micropayment and Free Content", http://shirky.com/writings/fame_vs_fortune.html, September 5, 2003.

[12] Yang, Beverly and Garcia-Molina, Hector "Emerging applications: PPay: micropayment for peer-to-peer systems" Proceedings of the 10th ACM conference on Computer and communication security, October 2003.

[13] Dai, J., QuickPay: and online payment protocol, Masters Report, Department of Computer Science, San Jose State University, 2006.

Sharing Application Logic across Programming Language Boundaries

Dennis S. Patrone Bina Ramamurthy
Department of Computer Science and Engineering, University at Buffalo
{dpatrone, bina}@cse.buffalo.edu

Abstract – As network-enabled devices and computerized services become ubiquitous in a breadth of applications, it is imperative that distributed system architectures enable components to communicate effectively and efficiently in heterogeneous environments. Standardized application-level network protocols are currently the most popular communication solution but a significant amount of flexibility is then lost at the network boundary. We propose an interaction model for distributed systems that will allow application knowledge in the form of programming logic to be shared across platform and programming language boundaries. We define an XML-based meta-language for describing application logic which is not targeted toward any one specific programming language, environment, or machine stack (real or virtual). Distributed applications can use this specification to exchange logic such as the details of application-level protocols in heterogeneous environments at runtime. This removes the distributed system's dependency on standardized and generalized application-level protocols. Distributed systems can also exchange algorithmic solutions to specific problems (knowledge dissemination), removing the need for any network-based communications to a centralized server. These freedoms will ultimately enable a distributed system that is more efficient, scalable, and adaptable than the current state of the art.

1. OVERVIEW

Distributed system components need to be able to communicate effectively and efficiently with each other regardless of their respective hardware platforms, operating systems, and programming languages (henceforth referred to in this paper simply as a system's "environment"). Current state-of-the-art in distributed system architectures provides only a partial solution. These systems generally achieve environment-independence by defining standardized, over-the-wire protocols that are targeted toward data sharing and request/response communications. Any environment can communicate with any other supported environment across network boundaries using the chosen standardized protocol. However, a significant amount of computational expressiveness is lost from the original programming language due to the inability of systems to share application logic across these language boundaries.

We propose an architecture which will allow distributed system participants to share not only data and method invocation requests, but actual application logic among heterogeneous components. We define an Extensible Markup Language (XML)-based meta-language, called the Application Logic Markup Language (ALML), which will allow distributed applications to share application knowledge in the form of data manipulation and flow control across language boundaries. The logic shared can include service-defined, client-side execution logic to create and modify application-level network protocols, allowing services to tune application-level protocols at runtime based on system-wide concerns such as usage patterns, network stability, and system loads. We hypothesize that a distributed system built on an ALML foundation will provide the basis for a more efficient, scalable, and adaptable distributed system than is possible using current state-of-the-art in platform- and language-independent distributed system technologies.

2. MOTIVATION

Current distributed technologies can be categorized into two broad architecture paradigms: message passing architectures and mobile code architectures.

2.1 Message passing architectures

Message passing architectures (e.g., CORBA [1], Web Services [2]) define system interfaces and network protocols to share data and remote procedure calls among heterogeneous components. Since the network traffic is defined as part of the standard, these architectures are often environment-independent and highly interoperable. They achieve this independence and interoperability at the cost of network flexibility. Their protocols defined independently of specific applications, and are therefore generally unable to exploit knowledge about a specific system's usage patterns, data flow, or other critical information which could help it potentially operate more effectively.

2.2 Mobile code architectures

Mobile code architectures (e.g., Java's Remote Method Invocation (RMI) [3]) on the other hand allow executable code to be shared from one distributed network node to

another. This code can contain implementation details on how the sender is going to modify the application-level protocol to improve performance for a given situation. The code could also contain algorithms that instruct the receiver how to perform a given function itself. These architectures are generally homogeneous in nature, at least at the communication level. This homogeneity is required for both ends of the communication pipe to be able to interpret the instructions that are communicated. While significantly more flexible with respect to controlling network bandwidth consumption and overall system performance (e.g., CPU utilization, remote request latency, etc) than message-passing architectures, expecting anything but prototypes and the smallest of real-world systems to be homogeneous is unrealistic.

2.3 A hybrid approach

Our goal is to create a distributed architecture that is a hybrid of these two approaches. This will be accomplished by combining the environment-independence of the message passing architectures with the protocol-independence of the mobile code architectures. As a first step, we are proposing the ALML specification as a "virtual language" to capture the application logic generally encoded in language-specific source code. An ALML-based distributed system is an environment-independent architecture with many of the benefits of a mobile-code architecture. These benefits include the ability to change network protocols at runtime and to inject new "knowledge" into clients at runtime. The significant distinction from other mobile-code architectures is that the information shared across the network in ALML is defined in a higher-level conceptual description of application logic rather than code compiled for a specific (virtual) machine stack. This logic description enables the system developer's intent to be interpreted and executed on the client side in any environment. Ultimately, this enables a mobile code architecture which also is environment-independent.

3. RELATED WORK

The Jini Network Technology architecture exhibits many of the dynamic benefits ALML is striving to reach [4]. However, Jini achieves these benefits in a "virtual platform-dependent" way, relying on a Java Virtual Machine (JVM) to interpret the compiled, shared application logic defined in Java's bytecode. Even though Java and .NET's Common Language Runtime (CLR) are technically platform independent by themselves, they are defined as different virtual machines and are not portable across their virtual boundaries. That is, Java bytecode will not execute on the CLR, and .NET's Common Intermediate Language (CIL) will not execute on a JVM. By specifying and sharing logic in portable XML, ALML allows the same logic representation to be interpreted in both, as well as any other existing and yet-to-be-defined, environments.

Reference [5] describes the general use of XML in distributed systems: a method to share data across the network. The XML specification allows application-specific tags to be defined. These tags are used to qualify data for storage and transfer in an environment-independent way. However, XML is simply a file-format specification and therefore details such as how to get XML files from one component to another in a distributed system are not included. Also, the meaning of the XML tags are not generally codified in XML but rather specified externally. So while XML may help applications understand distributed data it does not solve the data distribution problem. ALML adds the architecture-defined tags, their meanings, and bootstrapping protocols necessary to enable a distributed logic-sharing system.

Reference [6] proposes using XML to describe the behavioral specification (preconditions and postconditions) of using distributed resources. This is one step toward allowing distributed systems to better self-organize, but still requires compile-time knowledge about the application wire protocol in order to share this information. This approach only describes what will happen; it does not allow the service to describe how it will happen. XML has been proposed as its own programming language (e.g., [7], [8]). These languages have been targeted at their own unique runtime environments. Our language is not targeted at any one runtime environment but is a formal way to capture application logic that can be interpreted and executed in any number of environments. Turning that logic into an executable process is left to the end-consumer, maintaining the portability across environments. XML has also been used to define purpose-specific languages for modeling various processes in different applications (e.g., [9], [10]). ALML is an attempt to develop a general-purpose language which can be used by any application to describe the necessary flow of control and application logic in order to achieve some goal. Rather than defining purpose-specific XML tags, ALML defines generalized programming structures and flow-control statements allowing distributed systems to develop their own specialized capabilities.

4. APPLICATION LOGIC MARKUP LANGUAGE

The ALML architecture, shown in figure 1, is built on top of existing (and yet-to-be-defined) environments. The ALML specification and its corresponding XML schema definition (XSD) define the ALML logic grammar. A wire protocol for ALML-aware clients to "find and bind" with ALML-enabled services is included. A library (defined in ALML) provides generally useful utilities to ALML clients.

And an ALML engine can be included to process and interpret ALML logic definitions on the client-side.

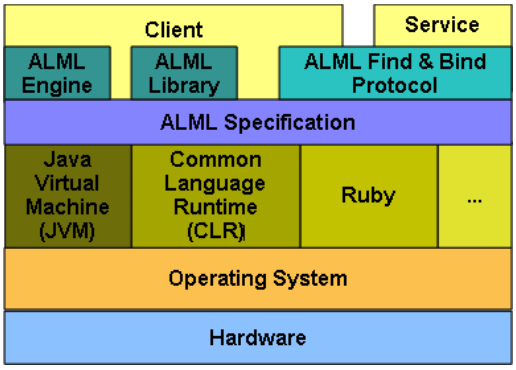


FIGURE 1: A HIGH-LEVEL VIEW OF THE ALML ARCHITECTURE.

While services can utilize ALML internally, this is not a requirement. Services need to provide an ALML-defined proxy via the find-and-bind protocol but need not execute the proxy logic at runtime themselves. The ALML proxy provided to the client will be interpreted in the client's environment, executing whatever application logic that service implementation defines.

4.1 The ALML specification

The ALML specification is an XML schema definition (XSD) which defines the ALML grammar and upon which the ALML system is built. The XSD includes standard object-oriented concepts including class definitions, packages or modules for grouping related classes, member variables, and methods. Standard flow-of-control constructs are also provided (e.g., looping, conditionals). The ALML specification is strongly typed, defining it own set of primitives. Standard visibility constructs are also provided.

4.2 The ALML find-and-bind protocol

The ALML distributed system will require a bootstrapping process which will allow clients to find desired services on the network and obtain the ALML-based proxy definitions at runtime without human intervention. While the specific details of this protocol have not yet been defined, it is expected that the specification will at a minimum provide multiple ways to locate a desired service such as the capabilities a client desires, definition of service attributes, and specific service addressing. The protocol should also operate over both UDP (broadcast for locating on a local network) and TCP ("well-known" locations for fixed installations and operation across routers and firewalls. Finally, the protocol will utilize widely accepted and supported application-level protocols (e.g., HTTP) and addressing mechanisms (e.g., URI) when appropriate.

4.3 The ALML library

The ALML library contains a set of classes that are guaranteed to be available to all ALML-compliant execution systems. This library provides useful capabilities that ALML services can make use of without implementing a significant amount of ALML code themselves and without requiring an extensive amount of XML to be downloaded to clients each time an ALML service is invoked. The library also provides hooks for ALML engines to provide certain capabilities which are tied to the specific client-side environments. The necessary library classes are still to be determined, but it is expected that standard mathematical operations (e.g., sin, cos, sqrt), networking objects (e.g., sockets), and standard input and output mechanisms will be a part of the toolset.

4.4 The ALML engine

An ALML engine is implemented for each supported environment. It is responsible for interpreting the XML-defined logic into environment-specific executable instructions at runtime.

5. LIZARD

A proof of concept system code-named Lizard is currently under development. The system implements core components of the ALML architecture in Java. It is being used to help identify required library classes and gaps in the ALML specification.

The proof-of-concept system is currently comprised of two parts. The first is the ALML XSD and a set of Java classes generated by JAX-B to operate over ALML XML files. The second is a specialized Java class loader that per request 1., converts a local ALML XML file into Java source file, 2., compiles the generated Java source file using the standard Java compiler javac, and 3., loads in the newly-generated class for client applications to use.

This particular implementation is not a requirement of an ALML engine; this is only a reference implementation. The extra step of generating source and compiling it using a Java compiler adds more overhead than having the ALML engine directly interpret or internally compile the ALML XML. However, this approach was chosen in the interest of saving development time. Presumably once the specification is well-defined, future implementations can focus more on runtime performance and less on ease-of-implementation.

As the system evolves, we plan to completely define the basic requirements for the ALML system and library. Currently only flow of control statements are supported. We also intend to define and implement a find-and-bind capability, including the ability to download ALML-related

files at runtime. Currently the system expects specific XML files in the local standard class path.

6. EFFICIENCY, SCALABILITY, AND ADAPTABILITY

We hypothesize that a distributed system architecture that supports sharing application logic will lead to systems that are more efficient, scalable, and adaptable than systems based on standardized network protocols. Distributed systems based on standardized network protocols by definition cannot modify network communications. These systems utilize compile-time bound client-side proxies—the code that communicates with the server over the network on the client's behalf. They are bound to the client at compile time since the proxy details are completely defined as part of the architecture's protocol standardization. Distributed systems that are capable of sharing application logic at runtime (mobile code) can exploit situational knowledge by deferring client-side proxy implementation details until runtime and changing them at any time. The details can then be tuned to any specific criteria which is important to the specific system (e.g., minimize network utilization, reduce latency, improve scalability).

6.1 Improving efficiency

Efficiency can be measured in a number of different ways: network throughput, CPU utilization, memory footprint, power consumption, etc. Every application will have its own criteria and thresholds for acceptable performance across scarce resources. Every protocol and usage pattern will affect different performance measurements uniquely. Selecting one generalized, standardized protocol will introduce a design bias and cannot possibly efficiently cover every situation. A system which allows services to provide mobile code to define smart proxies at runtime enables the system to react and adjust to changing situations.

For example, consider a simple banking distributed application with a Loan object that has three methods:

```
void makePayment(double amount);
double getBalance();
double getPayoffAmount();
```

Ignoring the complexities of exception handling and synchronization in distributed systems, we assume "makePayment" reduces the loan's principle by "amount" less any interest payments (also ignored here). The method "getBalance" return the loans current balance, or principle yet to be repaid. The method "getPayoffAmount()" returns the total amount to payoff the loan in one payment (assumed to be the principle possibly plus some interest and/or bank fees).

In defining an application-level network protocol, we must consider the expected usage of the Loan object. If we expect methods to be called individually and uniformly then a simple request-response messaging system per method may be appropriate. But if we expect the loan object to be utilized in a batch processing system where "makePayment" will be called n times (where $n > 1$) before a call to "getBalance" we may be able to significantly improve system performance by locally caching the total of all "makePayment" calls on the client side, and then making a single remote "makePayment" call to the service with the cached total as the first step in a "getBalance()" or "getPayoffAmount()" request. This level of batching will reduce network utilization by saving (n-1) "makePayment" requests being sent across the network. It will also reduce CPU utilization on the client by saving (n-1) marshalling and unmarshalling of remote requests. It further reduces the CPU service load by also distributing the logic for adding the payment (n-1) times to the client system. Finally, it reduces system latency by saving (n-1) network round-trips. And an even smart proxy could contain the logic for both proxy types, and chose the appropriate protocol based on usage patterns to even further improve overall system performance.

A mobile code-based architecture does not automatically solve the efficiency problem but rather provides a framework in which system designers and software engineers are able to solve distribution issues in a domain-specific and application-aware way.

There is extra cost associated with setting up a connection in a mobile code-based architecture, especially if that code is interpreted. There are additional up-front costs with respect to the network as the proxy code is downloaded. There are additional costs to CPU usage as the proxy code is compiled or interpreted on the first time it is used. There is also additional latency introduced in the initial call as the mobile definition is downloaded and interpreted. These costs, however, should be constant and therefore independent of n. Furthermore, in special circumstances system clients could also be seeded with initial proxies to further reduce initial connection costs when required.

6.2 Improving scalability

As discussed in section 6.1, tuning application-level protocols based on system usage can reduce demands placed on specific resources and/or distribute the processing of a distributed system, allowing the same software and hardware to provide greater capacity. Even so, if that capacity is reached, mobile code-based smart proxies can continue to help by providing the ability to redirect entire services, individual methods, and even specific requests across multiple servers helping systems achieve scalability.

These redirections can be based on past performance, current situations, or overall system goals—and all managed internally to the smart proxy on the client, but without tight client coupling. In some instances, the smart proxies can contain the entire logic to perform algorithmic-based services, further distributing the execution of a service across client machines, but maintaining the algorithm implementation definition at the service. If the algorithm needs to be updated in the future, the single service implementation is updated and the new algorithmic solution will be automatically pushed to all clients. This maintains the service distribution while helping with system maintenance.

6.3 Improving adaptability

As detailed in section 6.1, a mobile code-based architecture has the potential to enable smart proxies. These smart proxies, in turn, enable a distributed system to adjust runtime, application-level protocols to changing system demands. Section 6.2 describes how a mobile code-based architecture can also aid in adaptability when a service method or set of methods have algorithmic solutions. The algorithmic solution can be provided in the proxy, removing the need for any client-side communication back to the original service. The service is in effect "teaching" clients how to perform the service for themselves. This is very adaptable as clients can learn new "skills", and services can automatically update clients with improved algorithmic solutions, code the manage additional requirements or to handle emerging issues by changing the proxy implementation being provided.

7. CONCLUSIONS AND FUTURE WORK

We have proposed the Application Logic Markup Language (ALML) XML-based specification as a method for distributed applications to share application knowledge in the form of data manipulation and flow control across programming language boundaries. Our motivation for this work is to create a distributed system architecture which is capable of creating more efficient, scalable, and adaptable distributed systems than architectures based on standardized network-level protocols can. We have developed a proof-of-concept to show that describing application logic with an XML schema and executing that logic within a specific environment is possible. This is currently a work in progress. We need to completely specify the ALML language and determine if and how ALML can take advantage of environment-specific libraries and target platforms' native capabilities. A find-and-bind protocol to allow distributed systems to configure at runtime must also be defined. With mobile code comes great security concerns and these must also be addressed. Overall efficiency of the ALML specification must also be considered. For example a versioning mechanism for ALML documents may be appropriate in order to allow ALML engines to reduce redundancy when downloading and interpreting ALML files. Finally, we must provide ALML engine implementations in several diverse languages to show the portability and efficacy of ALML.

8. REFERENCES

[1] Object Management Group. CORBA Specification, Available at: http://www.omg.org/spec/CORBA/3.1/. January 2008. Accessed on: February 11, 2008.

[2] Booth, D. Haas, H., McCabe, F., Newcomer, E., Champion, M., et al. Web Services Architecture. 11 February 2004. Available at: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. Accessed on: February 11, 2008.

[3] Sun Microsystems, Remote Method Invocation Specification, Available at: http://java.sun.com/javase/6/docs/platform/rmi/spec/rmiTOC.html. Accessed on: February 11, 2008.

[4] Waldo, J. The End of Protocols, Available at: http://java.sun.com/developer/technicalArticles/jini/protocols.html. Accessed on: February 11, 2008.

[5] Deadman, R. "XML as a Distributed Application Protocol". Java Report, 1999, 4, 16-21.

[6] Mckee, P. and Marshall, I. "Behavioural specification using XML". Distributed Computing Systems, 1999. Proceedings. 7th IEEE workshop on Future Trends of, 1999, 53-59.

[7] Plusch, M. and Fry, C. Water: Simplified Web Services and XML Programming. 1. John Wiley & Sons, 2003.

[8] Klang, M. "XML and the art of code maintenance". Extreme Markup Languages. Proceedings of, 2003.

[9] Jordan, D, Evdemon, J (chairs), et al. Web Services Business Process Execution Language Version 2.0. 11 April 2007. Available at: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf. Accessed on: February 11, 2008.

[10] Workflow Management Coalition. Process Definition Interface – XML Process Definition Language. WFMC-TC-1025. Version 2.0. October 3, 2005.

Synergizing Collaboration and Reuse in Software Engineering

Stefan Seedorf*) and Oliver Hummel†)

University of Mannheim
Chair in Information Systems III), Chair of Software Engineering†)*
D-68131 Mannheim, Germany
seedorf@wifo3.uni-mannheim.de, hummel@informatik.uni-mannheim.de

***Abstract*— Reuse is widely considered key to improving software productivity and quality. It appears in various forms – ranging from independent binary components to highly abstract knowledge codified as design patterns. Common to all these approaches is that they involve different people working together to construct reusable assets and adapt them to work in various contexts. In that sense, reuse is impossible to achieve without collaboration. In this paper, we therefore claim that collaboration should receive a higher profile in reuse. First, we investigate the collaborative aspects in two selected reuse approaches. We then show how they can benefit from collaborative tools, i.e. wikis and collaborative development environments, and propose a novel integration of the latter with recently developed component retrieval systems.**

I. Introduction

Software reuse has been assigned a pivotal role in substantially improving development cost, software quality and time-to-market [1]. With the vision of software reuse solving the "software crisis", many new development paradigms, methods and tools have been developed over the years. Nearly four decades after Software Engineering emerged as a discipline, reuse has retained most of its relevance in both academia and industry. Reuse approaches have been successfully adopted in a number of instances, e.g. Component-based Development (CBD) [2] and design patterns [3]. However, its interpretation has gradually shifted towards embracing any kind of knowledge that is reused during the development process [4, 5].

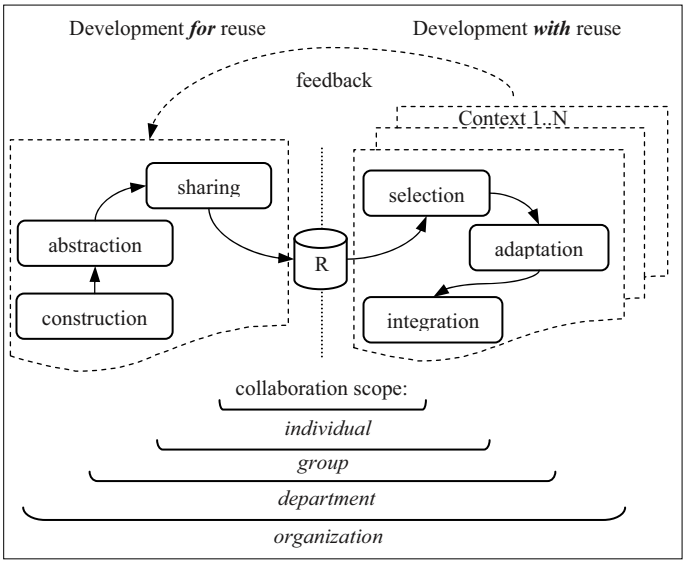
Although it has long been recognized that reuse is not only a technical but also a cognitive and social problem [6], the collaborative character of reuse has remained largely uncovered. Software development processes have recently become increasingly distributed since often geographically dispersed teams have to work together on large-scale development projects [7]. That is why issues arising with collaborative software development (CSD) have started to play a key role in software engineering research [8]. Collaboration is particularly relevant in the field of software reuse because it typically involves different stakeholders working together horizontally, by crossing organizational boundaries, and vertically, by cutting through different aspects of software engineering processes.

In this paper, we therefore investigate how software reuse approaches can be leveraged by taking a collaboration perspective. The paper is separated into two main parts: In section two, we study the fundamental relationship between collaboration and reuse. We analyze two cases in order to better understand the role of collaboration. In section three, we then show how collaborative technologies can be applied to improve recent component reuse technologies. Finally, we conclude by summarizing our findings and identifying directions for future work.

II. Relationship between Collaboration and Reuse

Systematic reuse is commonly split up into the two dimensions development *for* reuse and development *with* reuse [9]. Development for reuse deals with the creation of reusable artifacts, while development with reuse deals with their utilization in other contexts. The generic process steps included in these dimensions are depicted in Figure II.1. First, a software artifact is initially constructed with or without the intention of reusing it in other contexts. However, in order to actually make an artifact reusable there is always some kind of abstraction involved [10, 11]. Subsequently, the artifact is privately or publicly shared, e.g. by publishing it in a repository. This will enable other people to locate the artifact as a candidate for reuse. Development with reuse involves the selection of a suitable artifact, which is subsequently adapted to meet the requirements of a specific context and integrated into a software system [10].

Collaboration between individuals, groups, departments and organizations can be observed within the reuse phases as well as at the transitions between them. In most successful reuse scenarios, a dynamic interaction between development for and with reuse can be observed. In order to examine this relationship in more detail we take a closer look at collaboration in two particular cases: First, we analyze design patterns as a highly



II.1 Reuse process

successful example of knowledge reuse. Second, we investigate the collaborative aspects of product-line engineering.

A. Case 1: Design Pattern Reuse

Background. Inspired by work of Christopher Alexander in civil architecture [3], design patterns emerged in the early 1990s as a highly successful technique for reusing software design and architectural knowledge. A small group in the object-orientation community came up with the idea of composing a catalogue of recurring designs that would assist developers in writing object-oriented software. The first systematic collection of design patterns was published in the popular "Gang-of-four" book [3]. Today patterns are one of the most prominent examples for high-quality knowledge reuse. A pattern describes a solution to a common design problem [12]. Recurring experience about a domain is distilled and captured in a way that makes this knowledge easily acquirable by someone who does not have it [13]. A pattern description consists of several elements: a pattern name, a description of a problem, a solution, a context in which the solution works and discussion of the consequences. It is usually documented in text form, and optionally enhanced by diagrams and sample code. Hence, a pattern is not a software artifact that can be reused "as is", instead always needs to be located, assessed and interpreted by one or more developers before it can be applied to a specific context.

Reuse process. Although patterns should not be seen as "ready-to-use" recipes, they allow large scale reuse of development experience. A design pattern may capture the experience of just one developer; nevertheless it can be applied millions of times [14]. The conception process follows a typical scheme: Since designing software is a highly complex activity, developers use abstractions to decompose the problem into smaller, more manageable parts. The same or similar design decisions are made for recurring problems. Since some of the designs turn out to be better than others, learning processes are triggered. While a developer uses reflection-in-action and gains more experience, she is able to make tacit knowledge about "good designs" explicit.

Once this kind of knowledge is conceptualized and documented in the form of a pattern, it can be shared with other developers. This can be done by publishing pattern catalogues in books or - in a more participatory style – using Wiki systems such as [15]. If other developers encounter a design problem, they can search these media to investigate if a similar problem has been solved earlier and been described as a pattern solution. Usually, developers are able to narrow down a list of patterns to a few interesting candidates. After carefully weighing the costs and benefits of these candidates, one is selected and applied to the problem at hand.

More experienced developers will be able to recall suitable patterns and instantly decide about their suitability. Since they have embodied the necessary knowledge, they can effectively apply a wide range of patterns in various contexts. As part of a feedback loop, the experiences made by applying patterns can also be made explicit to create more specialized patterns or to guide fellow developers.

Role of collaboration. Pattern reuse is characterized by collaboration at different stages. First, a new pattern needs to be conceptualized by one or more developers. Despite the fact that a single developer may have the initial idea and knowledge required to express a solution in the form of a pattern, it is likely that it emerges when interacting with other developers who encounter similar problems. A pattern is typically the result of ongoing discussions about "good design solutions" to reoccurring problems. In that sense, patterns capture collective developer experience.

When a solution has repeatedly proven successful in several cases, it is worth describing and publishing as a pattern. Patterns can be shared by work teams and departments, but also by larger communities. An example for the latter is the first Web-based wiki, also known as Portland Pattern Repository, which was constructed by Ward Cunningham in 1995 to allow patterns to be collaboratively edited by an online community [15]. Unlike choosing books or online repositories to publish patterns, wiki systems provide a highly interactive approach to knowledge exchange. Not only are developers with the same interests invited to contribute patterns, it is also much easier to enrich the knowledge base. Other developers can discuss the pros and cons, provide links to related patterns or add examples in other programming languages. Such interactive ways of collaboration across organizational boundaries during the construction and abstraction phase provide a powerful example for a Community-of-Practice [16].

However, sharing a pattern is only one side of the reuse story, because it needs to be identified and interpreted before it can be successfully applied. Developers need background knowledge to understand a pattern, e.g. in object-orientation and modeling notations such as the UML, and they have to understand the problem space to decide about its suitability. Just as design is seen as a collaborative effort, so is the decision process of selecting and applying patterns. To this end, patterns become a focal point for both individual and collective learning. This leads us towards a key role of collaboration in pattern reuse: "Not only do patterns teach useful techniques, they help people communicate better and reason about what they do and why" [14].

To summarize, we can distinguish two major forms of collaboration in pattern reuse: on the one hand, they provide a scalable collaborative approach to transferring their know-how across projects, departments and organizations. On the other hand, using a common language during design supports developers in communicating architectural and design knowledge. This allows developers to collaborate more effectively in critical stages of the development process.

B. Case 2: Product Line Engineering

Background. Many software developing organizations create "a family of similar, but slightly different, systems rather than a single system" [17]. There is a potential to reuse the overlapping functionality of these systems, which cannot not be tapped when developing system by system individually. Software product lines (or system families) were thus introduced to plan software reuse right from the start. A software product line is a set of similar software systems which are developed and maintained together [18]. It typically consists of a product line architecture, a set of software components and a set of products [19].

On the one hand, developing a product line can only be useful when the systems have at least some features in common. This is typically the case when choosing systems from the same

application domain. The commonalities of systems therefore constitute the skeleton of a product line. On the other hand, there are also variabilities that need to be taken into account. Variable features are incorporated into the software assets to make them customizable for individual systems. In Product Line Engineering (PLE) it is therefore crucial to set the correct scope of a system family and to plan the commonalities and variabilities ahead so that the necessity of subsequent changes is diminished. Hence, PLE has a deep impact on all phases of Software Engineering processes, starting with requirements management. A number of methodologies for PLE have been proposed, such as Foda [20], PuLSE [21], and KobrA [17].

Reuse process. PLE is separated into two independent phases: First, domain engineering deals with identifying the commonalities and variabilities of the scoped products. A domain-specific architecture with reusable components is developed. Second, application engineering is the process of developing individual products on the basis of a product line architecture. Domain engineering is development for reuse, while application engineering is development with reuse.

Unlike in individual systems development, the requirements of a product family have to be collectively identified and analyzed. Some of the products may already exist, others may be planned in future. This makes the definition of a product line's scope difficult [21]. It is thus recommended that everything included by a concrete product is part of the domain [22]. During domain analysis a reference model is built from the concepts and relationships in the scoped domain [23]. Together with the commonalities and variabilities it frames the overall product line architecture. The architecture is decomposed into reusable parts that will be integrated into the individual products. All reusable parts follow a generic design by capturing variability in the domain. All reusable artifacts and documentation are made available through a mutual component repository [23]. In application engineering, only requirements of the dedicated product are relevant. In the ideal case all artifacts have been constructed during the domain engineering phase and can be taken from the repository [23]. In order to meet the individual product requirements, the generic artifacts are customized according to the previously defined variation points. However, the reuse process is not as straightforward in reality, because product requirements are constantly evolving [24]. In an evolutionary approach, the ability to provide feedback from individual product development to product line development is thus crucial.

Role of collaboration. Developing a software product line is always a joint effort which involves stakeholders from different organizational entities as well as disciplines. The collaborative character of PLE, however, has only recently been stressed [25, 26]. In a systematic approach the product line scope and supported features have to be negotiated previous to constructing or reengineering reusable artifacts. Since the required business and technical knowledge is distributed among many participants, a mutual agreement has to be reached first. To this end, "a collaborative approach can facilitate understanding the different stakeholder concerns and converging on mutually acceptable solutions" [26]. EasyWinWin [27] is proposed as an interactive negotiation technique for reaching satisfactory agreement among the stakeholders. Features and domains are prioritized according to the expected costs and benefits. It is then decided what becomes part of the reuse infrastructure.

Later, domain and application engineering activities have to be coordinated so that double work is avoided. On the technical side, the challenges are addressed by component-based approaches such as KobrA [17]. Precise specifications make it possible to develop components by independent providers and reassemble them during application engineering. Splitting the software lifecycle into domain and application engineering requires feedback between the two phases – changes in the latter have to be communicated to the domain layer. If these aspects of PLE are not carefully considered, it is possible that the initiative fails to bring the promised benefits.

C. Discussion

In the previous subsections, we analyzed the role of collaboration in two reuse approaches that appear very different at first glance. Design patterns are a leading example of reusing highly distilled knowledge in the development process. PLE, on the other hand, deals with systematically planning the reuse of software artifacts in a family of similar products. In the first case, software development is primarily seen as a creative, agile process. The collaboration between producers and consumers can be characterized as loosely coupled, because it is not necessary for consumers to provide feedback. Nevertheless advancements in design patterns are mainly driven by open communities who contribute to the overall knowledge base and help to guide other developers.

In contrast, the second case highlights the engineering perspective on software development. Here, collaboration between domain engineering and application engineering occurs not only during the inception phase. Since the requirements are continuously evolving, changes in individual applications have to be propagated back to the domain level, driving the evolution of the product line and thus naturally require a tight integration of collaboration techniques.

At second glance, however, both approaches expose similarities in terms of the underlying knowledge processes. Reuse can be interpreted as a collaborative knowledge construction process where knowledge is not seen as a commodity to be consumed, e.g. by retrieving artifacts from a reuse repository, but something that is collaboratively designed, constructed [28] and consumed. Knowledge about the application domain, architecture or design does not reside in one person's head but is distributed across disciplines, work teams or organizational entities. Therefore, the reuse process has to be viewed as a process of knowledge formation, where a shared mental model is created and transformed to match an artifact's varying contexts. The challenge lies in creating an environment in which the underlying knowledge processes such as externalization and combination are effectively supported (cf. [29, 30]). To this end, communities-of-Practice [16] provide flexible and informal ways of collaboration in both design pattern reuse and PLE. They encourage people with the same interest to engage and interact in a group, which facilitates knowledge-sharing and social learning.

After characterizing software reuse as a collaborative process, we will now look at how collaborative technologies can be enhanced to improve current reuse practices.

III. Enhancing Collaborative Technologies to Support Reuse

In section two we argued that reuse is an inherently collaborative process. While this insight may appear apparent to many reuse researchers and practitioners, it nevertheless has a profound impact for the emergence of new reuse methods and tools. We therefore introduce two key collaborative technologies in software engineering and describe by example how they can be enhanced to support core reuse tasks.

A. Wikis

In recent years, wikis have emerged as one of the most successful collaborative technologies [31]. Wikis enable Web content to be simultaneously edited by multiple users. One of the core principles is to encourage every reader of a wiki page to become an author as well – simply by editing the page content using a wiki language. Because of the low entry barriers communities have now got the opportunity to asynchronously work on a common subject of interest. Nowadays the most successful example of wikis is the Wikipedia. Wikis have also been widely adopted for the documentation of software projects by both open-source communities and enterprises.

Hence, wikis are an interesting candidate for addressing collaborative issues in software reuse. Interestingly, the very first wiki engine was developed for the shared development of design patterns (see Sec.2.1). In most cases, however, wikis are solely employed as a project documentation tool supporting the accessibility of unstructured knowledge. For example, wikis often serve as a simple problem/solution database. Only recently, has the wiki paradigm been extended to realize more sophisticated applications. These extensions already address specific reuse problems or could easily be easily customized to do so. Advanced usage scenarios of wikis in software and knowledge reuse include:

1. Requirements engineering in PLE
2. Semantic matching of software components
3. Reusing knowledge models across software projects

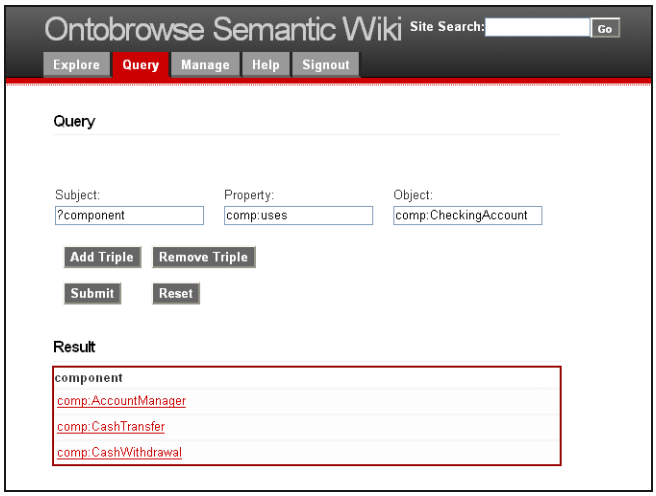
We will now discuss these advanced applications of wikis in more detail.

Requirements Engineering in PLE. Wikis have already been proposed as a lightweight approach for requirements elicitation, analysis, specification and management. The DisIRE (Distributed Internet Based Requirements Engineering) method employs Wikis for the distributed management of requirements [32]. The wiki pages are structured in a way that a requirement specification can be processed by external tools, e.g. to carry out a cost-benefit analysis. DisIRE can be applied to support the PLE scenario where the requirements of several products have to be managed as one (see Sec. 2.2). This is due to the fact that PLE is always a distributed process where the requirements need to be negotiated by multiple stakeholders. Wikis are also an appropriate tool for managing ongoing requirement changes, which is indispensable for long-term projects such as PLE.

Semantic component matching. Component matching is not an obvious feature of wikis. Nevertheless, recent semantic extensions have made it possible to utilize wikis as collaborative databases with machine-interpretable knowledge. Instead of purely processing text information as in traditional wikis, page content can now be decorated with semantic metadata. In this case, a knowledge model can be automatically extracted. This new generation of "semantic wikis" enables new applications in software engineering.

Ontobrowse semantic wiki has been specifically developed to support the sharing of architectural knowledge [33]. Pages are interpreted as entities, i.e. concepts, relations and objects. The wiki is thus no longer a "small web of formatted text pages" but a knowledge base that can be automatically processed. In Ontobrowse, a knowledge base is clearly separated into two parts: a knowledge structure defined by one or more ontologies (concepts and relations in a domain of interest) and instance knowledge defined by individual objects with their property descriptions. It is furthermore possible to add and modify text descriptions for every page as in traditional wikis.

In order to apply Ontobrowse to component matching, one has to set up the wiki in two steps. First, an ontology for software components has to be specified. It defines a terminology for describing software components, e.g. "component", "interface", "method", "input", "output" and "business object". Second, a plugin is created which maps component descriptions in a specific format (e.g. Java enterprise beans) to the ontology. The wiki can then be configured to automatically crawl for component descriptions in this format, extract the content and fill the wiki's knowledge base. Once the knowledge base has been initialized it is possible to formulate a specification-based query, e.g. for all components that use the business object "CheckingAccount" (see Figure III.1).



III.1 A semantic component query in Ontobrowse

The wiki provides a user-friendly frontend for specifying component queries and navigating component descriptions. Once a suitable component has been identified it can be easily retrieved if its location was inserted into the knowledge base.

Reusing knowledge models. In semantic wikis such as Ontobrowse it is not only possible but also encouraged to reuse the knowledge structure. Software projects setting up another wiki instance are able to reuse existing knowledge models (ontologies). Thus, reuse at the knowledge level is achieved.

Moreover, other knowledge models covering additional aspects of a software project – such as requirements and project management – can be added and refined as soon as they are required for a specific task.

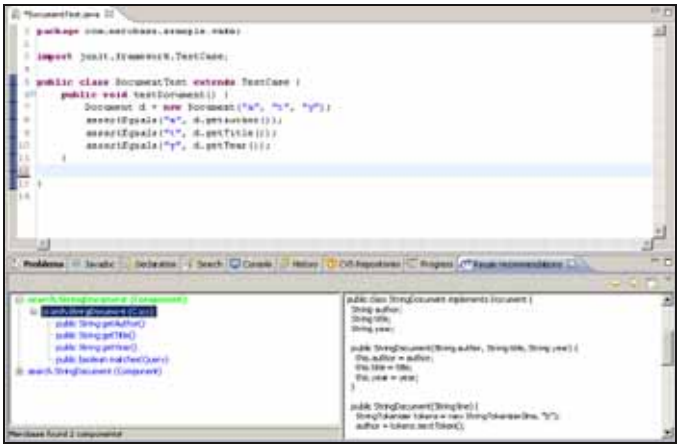
B. Collaborative Development Environments

In recent years, Collaborative Development Environments (CDEs) have appeared as a new category of development tools focusing not on individual productivity but supporting collaborative activities appropriately [34]. Typical features of a CDE are a versioned code repository for multi-side development, an issue tracker, project management and analysis tools, requirement management and documentation tools (e.g. wiki engine). Most platforms provide a Web interface as well as plugins for Integrated Development Environements (IDEs). Prominent examples of CDEs include CollabNet/Sourceforge[1], Codebeamer[2] and GForge[3].

Although a CDE allows many software projects to be managed in parallel, the potential for reuse across projects has yet to be recognized. So far CDEs provide little support for collaborative reuse of software components and code since common configuration management software such as CVS or SVN does not include support for targeted retrieval of reusable assets. In other words, search possibilities are limited and do not go beyond simple keyword matching even if a CDE includes source code in its searching capabilities. An additional problem is often the lack of awareness about potentially reusable software in large and distributed organizations. Thus, the challenge is to enhance CDEs with more sophisticated retrieval techniques for reusable material, i.e. source or binary components.

Although [35] provided a good overview on component retrieval techniques, the authors realized that none of them was actually usable in practice. To date, it is also not clear what the best approach for reusing components might be. Component-based development approaches such as KobrA [17] propose component selection based on a specification taken from the design of a system. This, however, requires retrieval techniques for components that go far beyond simple keyword matching. Another aspect related to this is certainly the problem of filling a repository with more than just a few hundred reusable components [36] since historically companies were normally not willing to open their intellectual property to researchers working on this challenge. Thus, the retrieval techniques for older prototypes were mostly too imprecise and not able to deal with the large numbers of software (typically many thousands if not millions of files) contained in the repositories of distributed companies [37] today. Up until now it is also not clear whether labor intensive approaches such as the previously mentioned semantic wikis or semantic web services will scale for that large number of reusable software.

In order to increase awareness of reusable components it is furthermore desirable that developers in one project are informed about similar software components in other projects without actively searching for them. This leads us to the idea of so-called proactive reuse recommendation tools that have first been popularized by Ye's CodeBroker [38] in the late 1990s. The basic idea is that a developer does not need to invest extra effort into querying the component repository – which can quickly lead to developers not reusing anymore if a few attempts have failed, according to [39] – since the reuse tool proactively searches for reusable material in the background and is supposed to present only those results that are likely to fit into the context a developer is working on. Thus, we believe a high relevance of the reuse results is crucial in this context. In an agile context the so-called Extreme Harvesting approach [40] utilizing test-driven reuse is able to deliver very precise reuse recommendations [37] based on queries that can be automatically derived from test cases as they are commonly used in agile development processes. Recently, the main effort was to develop a test-driven reuse tool tightly integrated into the Eclipse IDE in order to achieve fully proactive reuse recommendations for developers. Its test-driven mode requires a developer to apply an agile test-first development approach where our tool is able to recommend reusable candidates based on (JUnit) test cases a developer has just created. These test cases contain enough syntactical and semantical information to facilitate very precise recommendations, especially if a potential candidate could have been automatically tested in a secured (server-side) environment as shown in the following figure.



III.2 Screenshot of test-driven reuse in Eclipse

Alternatively, the developer is designing or coding as normal in his Eclipse environment. Meanwhile the recommendation tool is automatically analyzing the structure of the class under development and derives queries for the underlying component repository. Potential candidates (based on a syntax analysis) are immediately returned and proposed to the developer.

These features have been implemented in the Merobase search engine[4]. Its Eclipse plugin for providing proactive recommendations is called Code Conjurer[5]. The integration with CDEs is simple because Merobase is able to automatically build a unified index from a potentially large number of software configuration management repositories such as CVS or SVN from various company sites or even beyond company borders. The recently finished version can thus be used within the scope of collaborative development platforms. However, since it is not yet clear, how metadata such as UML diagram etc. from a wiki can

[1] http://www.collab.net
[2] http://www.intland.com
[3] http://gforge.org

[4] http://merobase.com
[5] http://merobase.com/plugin-manual.do

be integrated with the component retrieval server, we plan to present a more detailed discussion on this at another occasion.

IV. Conclusion

In this paper, we have taken a collaborative perspective on software and knowledge reuse. So far there has been very little emphasis on its collaborative character. In order to clarify the role of collaboration we have therefore analyzed two distinct reuse approaches, namely design patterns and PLE. Design patterns are a case of knowledge reuse, emphasizing the agile and community-based perspective towards software development. They enable design experience to be more easily transferred to other developers. PLE, on the other hand, reflects the systematic, engineering perspective. However, we have discovered that collaboration plays a vital role in PLE as well. Especially, community-based approaches are highly suited to support the continuous evolution of a software product line.

In the second part, we then investigated collaborative technologies that already receive a high profile in software development organizations and open source communities. In particular, software development wikis should receive an even higher attention. We have learned earlier that design pattern reuse inspired the development of wikis in the first place. In turn, one should also consider using wikis in other scenarios, e.g. to support requirements analysis for product families or matching reusable software components. In order to tap the full potential of software and knowledge reuse we call for a novel integration of recent reuse recommendation systems into collaborative environments. However, although the technical premises for semantic and pro-active component retrieval are already fulfilled, they still need to be integrated into CDEs. This will become our main focus in future work.

V. References

[1] J. Bosch, *Design and use of software architectures: adopting and evolving a product-line approach*: ACM Press/Addison-Wesley Publishing Co., 2000.

[2] C. Szyperski, *Component software*: Addison-Wesley Reading, Mass, 1999.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995.

[4] Y. Ye, "Supporting software development as knowledge-intensive and collaborative activity," *Proceedings of the 2006 international workshop on Workshop on interdisciplinary software engineering research*, pp. 15-22, 2006.

[5] V. R. Basili, G. Caldiera, and H. D. Rombach, "Experience Factory," *Encyclopedia of Software Engineering*, vol. 1, pp. 469-476, 1994.

[6] G. Fischer, "Cognitive View of Reuse and Redesign," *IEEE Software*, vol. 4, pp. 60-72, 1987.

[7] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter, "An Empirical Study of Global Software Development: Distance and Speed," *International Conference on Software Engineering*, pp. 15-18, 2001.

[8] T. Hildenbrand, F. Rothlauf, and A. Heinzl, "Ansätze zur kollaborativen Softwareerstellung," *Wirtschaftsinformatik*, vol. 49, pp. 72-80, 2007.

[9] I. Sommerville, *Software engineering*, 8th ed. Harlow, England; New York: Addison-Wesley, 2007.

[10] C. W. Krueger, "Software reuse," *ACM Comput. Surv.*, vol. 24, pp. 131-183, 1992.

[11] P. Wegner, "Varieties of Reusability," *Proc. ITT Workshop Reusability in Programming*, pp. 30-44, 1883.

[12] P. Stevens, "Software design patterns," *Computing & Control Engineering Journal*, vol. 11, pp. 160-162, 2000.

[13] R. Helm, "Patterns, architecture and software," *ACM SIGPLAN Notices*, vol. 31, pp. 2-3, 1996.

[14] D. Schmidt, "The Road to Reuse: Design Patterns," *Proceedings of the 4th International Conference on Software Reuse*, 1996.

[15] Portland Pattern Repository, "WikiWikiWeb: http://c2.com/cgi-bin/wiki," 2007.

[16] J. Lave and E. Wenger, *Situated Learning: legitimate peripheral participation*: Cambridge University Press, 1991.

[17] C. Atkinson, *Component based product line engineering with UML*. London; Munich: Addison-Wesley, 2002.

[18] P. Donohoe, *Software Product Lines: Experience and Research Directions*: Springer, 2000.

[19] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, H. Obbink, and K. Pohl, "Variability Issues in Software Product Lines," *Proceedings of the Fourth International Workshop on Product Family Engineering (PFE-4)*, pp. 11–19, 2001.

[20] K. C. Kang, *Feature-Oriented Domain Analysis (FODA) Feasibility Study*: Carnegie Mellon University, Software Engineering Institute, 1990.

[21] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J. M. DeBaud, "PuLSE: a methodology to develop software product lines," *Proceedings of the 1999 symposium on Software reusability*, pp. 122-131, 1999.

[22] J. M. DeBaud and K. Schmid, "A systematic approach to derive the scope of software product lines," *Proceedings of the 21st international conference on Software engineering*, pp. 34-43, 1999.

[23] J. Bayer, S. Kettemann, and D. Muthig, "Principles of Software Product Lines and Process Variants," PESOA-Report, 2004.

[24] P. Toft, D. Coleman, and J. Ohta, "A cooperative model for cross-divisional product development for a software product line," *Proceedings of the first conference on Software product lines: experience and research directions: experience and research directions table of contents*, pp. 111-132, 2000.

[25] T. E. Fægri, T. Dingsøyr, L. Jaccheri, P. Lago, and H. van Vliet, "Exploring Communities of Practice for Product Family Engineering," *LECTURE NOTES IN COMPUTER SCIENCE*, vol. 3782, pp. 96, 2005.

[26] M. A. Noor, R. Rabiser, and P. Grünbacher, "A Collaborative Approach for Reengineering-based Product Line Scoping," 2007.

[27] R. O. Briggs and P. Gruenbacher, "EasyWinWin: managing complexity in requirements negotiation with GSS," *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pp. 10, 2002.

[28] G. Fischer, "The Software Technology of the 21st Century: From Software Reuse to Collaborative Software Design", Proceedings of ISFST2001: International Symposium on Future Software Technology, ZhengZhou, China, 2001.

[29] O. Gendreau and P. N. Robillard, "Knowledge Conversion in Software Development", Proceedings of the Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2007), Boston, Massachusetts, USA, 2007.

[30] I. Nonaka and H. Takeuchi, *The knowledge-creating company: how Japanese companies create the dynamics of innovation*. New York: Oxford University Press, 1995.

[31] B. Leuf and W. Cunningham, *The Wiki way: quick collaboration on the Web*: Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2001.

[32] M. Geisser and T. Hildenbrand, "A Method for Collaborative Requirements Elicitation and Decision-Supported Requirements Analysis," 2006.

[33] H.-J. Happel and S. Seedorf, "Ontobrowse: A Semantic Wiki for Sharing Knowledge about Software Architectures", Proceedings of SEKE, 2007.

[34] G. Booch and A. Brown, "Collaborative Development Environments," *Advances in Computers*, vol. 59, pp. 1, 2003.

[35] A. Mili, R. Mili, and R. T. Mittermeir, "A survey of software reuse libraries," *Ann. Softw. Eng.*, vol. 5, pp. 349-414, 1998.

[36] R. C. Seacord, D. Plakosh, and G. A. Lewis, *Modernizing Legacy Systems: Software Technologies, Engineering Processes*: Addison-Wesley, 2003.

[37] O. Hummel, W. Janjic, and C. Atkinson, "Evaluating the Efficiency of Retrieval Methods for Component Repositories", Proceedings of SEKE, 2007.

[38] Y. Ye, "An Active and Adaptive Reuse Repository System", Proceedings of 34th Hawaii International Conference on System Sciences, 2001.

[39] W. B. Frakes and S. Isoda, "Success factors of systematic reuse," *Software, IEEE*, vol. 11, pp. 14, 1994.

[40] O. Hummel and C. Atkinson, "Extreme Harvesting: test-driven discovery and reuse of software components", Proceedings of the IEEE International Conference on Information Reuse and Integration, 2004.

Improving Component Container Development Process through Product Line Engineering

Guoliang Liu, Yang Li, Jun Wei
Technology Center of Software Engineering, Institute of Software,
Chinese Academy of Sciences,
P.O.Box 8718, Beijing 100080, P.R.China
{glliu, fallingboat, wj}@otcaix.iscas.ac.cn

Abstract

Component containers play a key role as the infrastructure of component-based distributed applications. Nowadays, various kinds of components are emerging to satisfy requirements of developing applications on Internet. Hence, it's becoming more and more important to improve development process of component containers with software reuse methods. Product line engineering has proven successful as systematical reuse method, and is the best choice for component container family comparing to other software reuse methods. However, diverse understandings of CBSD concepts and big difference among architecture of existing component containers issue great challenges to yield component container product line. In this paper, we first standardize basic concepts of CBSD, including component model and component container, along with their relations. Feature model of component containers is extracted based on these concepts. Then we present architectural design, including modules, interfaces and invocations, as well as commonality and variability analysis, which followed by product derivation process to produce component container with product line through several kinds of intuitive actions. Product line architecture and product derivation process comprise a component container product line, named PLACE. A case illustrating development process of a J2EE servlet container with PLACE shows enhancement of reusability and effectivity of the product line architecture.

1. Introduction

Component containers are cornerstone of development of component-based distributed applications, since they provide deployment and run-time infrastructure following given component models [2]. Container is responsible for creating and managing component instances, allocating system resources automatically, acting as interaction agent among components and interpreting remote requests. Functionalities provided to components are encapsulated as infrastructure services, such as transaction monitoring and logging, which are transparent to client users of components.

Along with increasing component types, even more (because of different versions of component specifications) component containers are required. However, most of them are not new, but variants of previous built systems, which makes it essential to reuse both design knowledge and existing code systematically among component containers.

Software reuse refers to the use of previously developed software resources in new applications. Because less development effort to be made and reusable software resources are rigorously tested, software reuse can increase productivity and software quality.

Most of current software reuse methods are inappropriate as principal guideline for systematical reuse of component container. Design pattern (including architectural pattern) and reuse libraries are excluded because of incomplete reusing scope, while containers are too complex and diverse for domain-specific reference architecture, framework and generative methods. Product line engineering has proven successful as systematical reuse method [3], and is the best choice.

Product line engineering has become an important and widely used approach for efficiently developing portfolios of software products. This approach

produces order-of-magnitude economic improvements compared to one-at-a-time software system development [3]. In product line engineering, software development process consists of two major processes or life cycles: software product line engineering and application engineering [12]. During software product line engineering process, the commonality and variability in the product line are analyzed in light of the overall requirements of the product line. During application engineering process, an individual application that is a member of the software product line is developed. Instead of starting from scratch, as is usually done with single systems, the application developers make full use of all the artifacts developed during the software product line engineering life cycle.

In next section, challenges to apply product line engineering to component container domain are analyzed. And in Section 3, a product line of component container, named PLACE, is presented by giving its two major parts: product line architecture and product derivation process. A case is given in Section 4, in which a J2EE servlet container is developed with PLACE, and shows improvement of reusability and effectivity of the product line architecture. After related works are compared in Section 5, Section 6 concludes this article and gives future works.

2. Challenges to Apply Product Line Engineering to Component Container Domain

The first challenge we are facing is requirement modeling. Sources of requirements of component container include component specifications and existing component containers, and all of them internally follow component-based software development (CBSD) theory. However, different explanations are adopted among component specifications [1][2][10]. In order to build a product line of component container, understanding of concepts from CBSD must be unified.

Heineman and Councill define component, component model and component model implementation (a.k.a. component container) and their relations in [2] as following, and as shown in Fig.1:

> **Definition 1** A *component* is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard. A *component model* defines specific interaction and composition standards, while a *component model implementation* supports the execution of components and their assemblies that conform to the model.

We will use Definition 1 as common understandings of these concepts in this paper, according to which component model will be used instead of component specification, standard or other terminology without confusing.

According to Definition 1, component model acts as behavior and development guidelines of components and component based software systems, which is also requirement source of component container. We then will use component model as base of feature modeling of component container, as shown in Section 3.1.

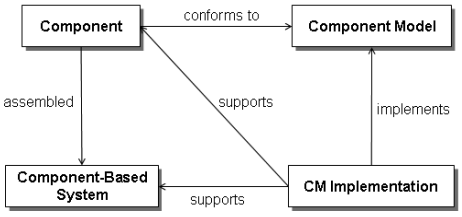


Fig.1 **Relationship among several concepts of CBSD**

The second challenge is abstraction of architectural design. Every component model has its own applicable area, for example EJB is middle-sized business component and web service is coarse-grained inter-organization component, because of what EJB using RMI/IIOP and JMS, supporting transaction and persistence while web service using SOAP along with WS-* specifications. Therefore, architectural design difference between component containers is critical for building product line.

Fortunately, basing on common understanding to CBSD concepts what we talked in the first challenge, component containers have common workflow (server/client paradigm and remote request handling), similar use cases (request handling, component deploying, system monitoring etc.), so that can be implemented by same set of architectural patterns and variability realizing techniques [4], which is of great help for handling design alternatives.

3. Product Line Architecture Design

3.1. Feature Modeling of Component Container

According to existing domain modeling methods, features are abstraction of services provided by and

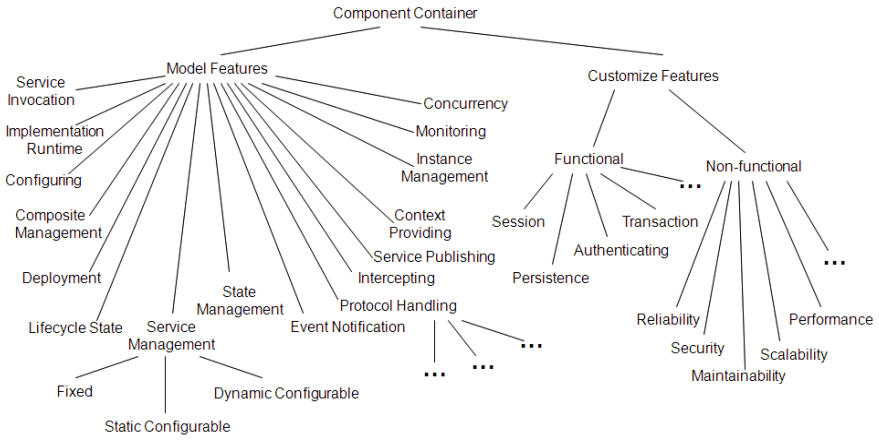


Fig.2 **Component Container Feature Graph**

techniques used in applications, and they are used by domain experts to communicate their idea, needs and problems. Feature-oriented modeling technique was first presented by Kang etc.[5]

Requirements of component container can be divided into two categories: those from component models and from container vendors. We call them model requirements and customized requirements respectively. Model requirements are explicitly regulated through component specifications, interface definition languages or protocol standards. Customized requirements are those except model requirement, which is proposed by container vendors basing on marketing strategy, user desire, development cost or state of practice. Features of product line are extracted from common requirements of most products in it. Therefore, features of component container consist of model features and customized features respectively. Layered relationships among features are shown in Fig.2, in which higher layer in the tree indicates more abstractive feature and lower layer means more concrete feature.

3.2. Architectural Design of Container Product Line – PLACE

In this section, we propose an architectural design of container product line, named PLACE (Product Line Architecture of component Container Environment), which is based on analysis in former sections as well as state-of-practice of architectural design of component containers. Modules and their relationship in PLACE are shown in Fig.3.

There are three kinds of architectural elements present in Fig.3. *Modules* are depicted by boxes, while *interfaces* are represented by black triangles on the edge of module boxes. Inner or outer directions of triangles show required interfaces or provided interfaces respectively. And *invocations* are by lines connecting pairs of interfaces. Modules, interfaces and invocations with dotted lines mean *optional elements*, while solid lines mean *mandatory elements*. Mandatory elements exist in architectures of all products derived from PLACE, comparing to optional elements which are only in architectures of some products within the product line.

We will introduce modules and their interrelations briefly.

Protocol processor is responsible for parsing and composing messages used to communicate with clients through network protocols. It converts request messages into in-memory objects, which are passed on to *service manager*; meanwhile it also composes response messages with results received from *service manager*, and sends them back to clients. Sometimes, p*rotocol processor* will invoke *concurrency controller* to handle requests concurrently.

Service manager is in charge of request dispatching as well as lifecycle managing of container services, which includes loading, starting and stopping. After *protocol processor* converts requests messages into objects, *service manager* processes them with intercepting container services, and then passes them on to *component manager*. Sometimes s*ervice manager* also invokes *concurrency controller* to provide container services with concurrency support, or send run-time status to *monitor*.

Component manager integrates many functionalities such as component invoking, configuring, lifecycle managing, lifecycle events notifying. *Deployer* invokes *component manager* when

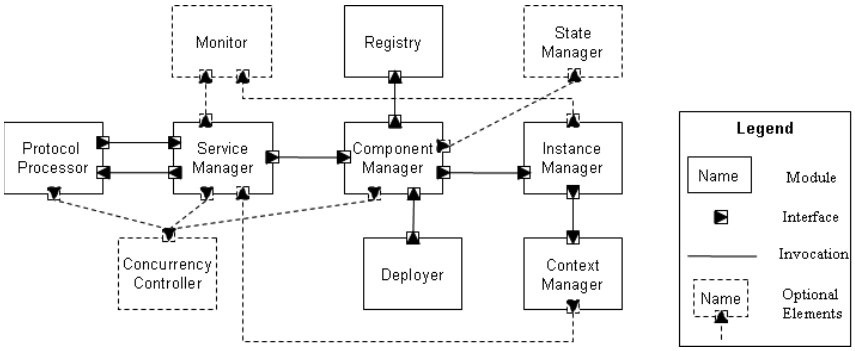


Fig.3 **Architectural Design of PLACE**

deploying components or component applications. When handling requests, *component manager* invokes *registry*, *state manager* and *instance manager* respectively, to look up, manage component states and component instances. *Component manager* also probably invoke *concurrent controller* to provide concurrency support to its submodules.

Besides providing component instances with environmental information, c*ontext manager* also receives and deals with invocations of run-time container services, which are perform by invoking *service manager*.

Monitor collects run-time information of container, whose possible sources are *component manager* and *instance manager* based on categories of information in current containers.

3.3. Commonalities and Variabilities in PLACE

Commonalities in PLACE include architectural modules' separation, relationships among them which defined by abstract interfaces and data structure of messages transformed among modules.

As shown in Section 3.2, these commonalities are based on common requirements of all component containers within PLACE according to feature graph in Section 3.1. For example, Deployer module is corresponding to 'deployment' feature in Fig.2, and its interface provided to Component Manger defines abstract methods as following. It accords to general actions of deployment functionality.

```
interface Deployer {
  boolean deploy(ComponentIdentifier compId)
            throws DeploymentException;
  boolean undeploy(ComponentIdentifier compId)
            throws DeploymentException;
}
```

Variabilities in PLACE can be classified into functional variability and non-functional variability. Functional variability includes adding or removing functionalities, adjusting behaviors of or relationships between modules, while non-functional variability includes extensibility etc.

Developer will realize variability by choosing one from design alternatives in product derivation process, which is explained in next section.

3.4. Product Derivation Process

Designing containers with PLACE has two steps: 1) product requirements analysis, during which architect compares concrete component model against common features in PLACE; 2) deriving concrete container architecture from PLACE, including three kinds of actions: *specialization, removal* and *augmentation*.

Specialization means to replace abstract modules supporting generic features with concrete modules supporting specific features. *Removal* means to remove modules or invocations not needed by concrete container, since requirements of every single container are only subset of that of PLACE. *Augmentation* is to add modules, interfaces or invocations implementing requirements that are not covered by PLACE.

4. Case Study

ONCE platform [5] consists of a series of software infrastructure products developed by Institute of Software, Chinese Academy of Sciences. Several representative component containers are included in ONCE, such as Servlet, EJB, web service, Portlet and BPEL containers.

Containers in ONCE have these following characteristics: a) Component models of all these containers are distributed component. b) For development of reusable assets, all products should use same programming language. Our choice is Java, since

most of them (EJB, Servlet, Portlet) belong to J2EE1 by Sun Microsystems, Inc., and the other two component models are also compatible with it.

We have applied PLACE in design practice of component containers in ONCE. Next, we will take design process of a Servlet container as a example.

Servlet technology2 resides in representative layer of three-layer architecture of J2EE applications, which is responsible for generating user interface code and handling interaction with clients. During development of ONCE products, we proposed following requirements to Servlet container to enhance usability:

a) Hot deployment, which means to deploy or undeploy Servlet applications (in WAR files) at run-time.

b) Web Administrative Console (WAC). User can monitor and manage components and component instances in Servlet container remotely through web user interface, and also can monitor components and instances by viewing various statistics information provided by WAC, start or stop deployed Servlet components. WAC should also support remote deploying, which allow client-side user to upload and deploy WAR files through web browser.

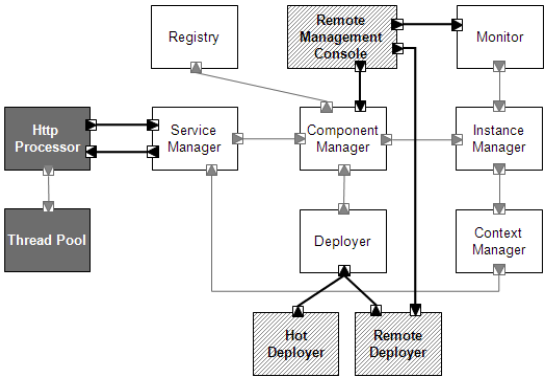


Fig.4 **Architecture of Servlet container derived from PLACE**

Comparing to PLACE architecture in Fig.3, Servlet container's architecture in Fig.4 shows the following variabilities:

Specialization. Protocol Processor and Concurrency Controller in PLACE are specialized to Http Processor and Thread Pool respectively, depicted by boxes with dark gray background. Protocol Processor follows monolithic design since this Servlet container only supports HTTP protocol, so it's specialized into Http Processor, and invocation parameters between Http Processor and Service Manager are OnceHttpRequest and OnceHttpResponse.

Removal. State Manager and related invocations are removed, so are invocations from Service Manager/Component Manager to Thread Pool, and from Monitor to Service Manager. Removed architectural elements are not in Fig.5 anymore.

Augmentation. Added parts include Remote Management Console, Hot Deployer, Remote Deployer and corresponding interfaces and invocations, which are shown as boxes with shadow background and bold lines

4.1. Discussion

In this case, only 3 of 12 modules are built from scratch, while 7 of 12 modules' definitions keep unchanged comparing to those in PLACE. Although development cost of different modules are very different and some implementing code may be written to complete modules' functionalities, total development time and cost is deduced remarkably.

While most of top-level modules being reused, reusability of source code is increased too. For example, thread pool which is originally developed for EJB container can be reused without rewritten except changing several parameters, such as size of thread pool.

5. Related Work

One research area related to our work is domain analysis [7] [8] [9]. Moon et al. [7] introduces a process for developing domain requirements where commonality and variability in a domain are explicitly considered. Within their metamodel for domain requirement, variation points are categorized into four types: computation, external computation, control, and data, which is similar to our classification of variations in component models. They also identify computation, external computation, and control variations in BehaviorPRelements (a perspective along a timeline), and data variations in StaticPRelements (a perspective along static structure). Chastek et al. [8] and Mei et al. [9] also propose interesting approaches to elicit and model domain requirement, but they all focus on requirement level of product lines, do not think about design and implementation levels as we do in PLACE.

Our work is also inspired by publications on component-based software engineering and component models [1] [2] [10]. Lau et al. [1] classify component

[1] Which also known as Java EE now. Java EE at a Glance, http://java.sun.com/javaee/

[2] Java Servlet Technology, http://java.sun.com/products/servlet/

models into a taxonomy based on commonly accepted desiderata for CBD (Component-Based Development) and evaluate categories with respect to these desiderata. Lau et al. regard software component model as a definition of semantics, syntax and composition of components. As we already quoted in Section 1, Heineman and Councill [2] define these concepts, component, component model and component model implementation, with each other. Szyperski et al. [10] define component without component model, and tend to look into component models and platforms from technical and strategic perspective. The main differences of our work are more detailed analysis and narrowed scope, based on container product line point of view. Their definitions of component models are too coarse-grained to guide domain analysis and architectural design.

6. Conclusion and Future Work

Software reuse can improve quality and productivity of component containers, which is crucial for fulfilling requirements of emerging component models. Product line engineering is a promising technique for systematical reuse of portfolios of software products. However, applying product line engineering is facing challenges including different understandings of basic concepts from CBSD and serious diversity among architectural designs of existing component containers.

In this paper, we analyzed these challenges and proposed PLACE: a product line of component containers, which consists of two major parts: product line architecture and product derivation process. Based on common understanding of basic concepts and feature modeling of component container, architectural design reflects general modules' separation, interfaces' definition and interacting relationships among them, while product derivation process shows practical procedure to design a component container product with PLACE with three kinds of intuitive actions. In the end, a case shows effectiveness of PLACE by illustrating development process of a Servlet container.

As future work, we are analyzing component models with variability modeling techniques, such as COVAMOF [11], to further reveal their interrelationship, and making effort to generalize this approach to apply to other product lines.

Acknowledgments. This work is partially supported by the National Natural Science Foundation of China under Grant No. 60573126; the National Basic Research Program of China (973) under Grant No. 2002CB312005; the National High-Tech R&D Plan of China (863) under Grant No. 2006AA01Z19B, 2006AA01Z180; the National Key Technology R&D Program of China under Grant No. 2006BAH02A01.

References:

[1] Lau, K.-K., Wang, Z.: Software Component Models. IEEE Transactions on Software Engineering, Vol.33, No.11 (2007) 709-724

[2] Heinemann, G.T., Councill, W.T.: Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley (2001)

[3] Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley (2002)

[4] Svahnberg, M., van Gurp, J., Bosch, J.: A taxonomy of variable realization techniques. Software Practice & Experience, Vol.35 (2005) 1–50

[5] Kang, K.C., Cohen, S., Hess, J., et al: Feature-Oriented Domain Analysis Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (1990)

[6] Institute of Software, Chinese Academy of Sciences: ONCE platform. http://www.once.com.cn

[7] Moon, M., Yeom, K.: An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in a Product Lines. IEEE Transactions on Software Engineering, Vol. 31, No. 07 (2005) 551-569

[8] Chastek, G., Donohoe, P., Kang, K., Thiel, S.: Product Line Analysis: A Practical Introduction. Technical Report CMU/SEI-2001-TR-001, Software Engineering Institute, Carnegie Mellon University (2001)

[9] Mei, H., Zhang, W., Gu, F.: A Feature Oriented Approach to Modeling and Reusing Requirements of Software Product Lines. Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC'03), IEEE Computer Society (2003) 250-256

[10] Szyperski, C., Gruntz, D., Murer, S.: Component Software: Beyond Object-Oriented Programming, second edition. Addison-Wesley (2002)

[11] Sinnema, M., Deelstra, S., Nijhuis, J., Bosch, J.: COVAMOF: A Framework for Modeling Variability in Software Product Families. Proceedings of 3rd International Software Product Lines Conference (SPLC 2004), LNCS 3154 (2004) 197-213

[12] Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley (2004)

.NET Extensions to the π-Architecture Description Language

Zawar Qayyum*, Flavio Oquendo
University of South Brittany
VALORIA – BP 573 – 56017 Vannes Cedex, France
zawar.qayyum@univ-ubs.fr, flavio.oquendo@univ-ubs.fr

Abstract – **The π-Architecture Description Language (π-ADL) is a high level software architecture specification language, formally founded on the π-Calculus. π-ADL.NET is a software implementation developed for the purpose of integrating π-ADL in the Microsoft .NET platform, with the view of expanding the usage horizon and application possibilities of π-ADL as an executable formal language for prototyping and rapid architecture-centric development.**

Since .NET is a multilingual development platform, the ability of a .NET language to access software components written in other .NET languages opens up new possibilities for multilingual software development using π-ADL.

This paper proposes extensions to π-ADL to be implemented in the π-ADL.NET compiler, in order to allow it to use existing .NET APIs written in other .NET languages. The proposed extensions cover all the features of the .NET language foundation: namespaces, classes, constructors, methods, fields, properties, events, enumerations, exceptions, null-ness, method delegation, different aspects of casting, generics and inner classes.

Index Terms – **π-Architecture Description Language; .NET; Compiler; Platform Specific Extensions**

I. INTRODUCTION

An Architecture Description Language (ADL) is a language especially conceived to address the modelling needs of software architectures. Various ADLs have been designed with different approaches to software architecture modelling [1]. π-ADL, a relatively recent ADL [2] adopts a formal approach towards software architecture specification, based on the π-calculus. The main advantage of this approach is the reliability of the resulting system, a very important area of focus for large and complex software. However, the software architecture description should also be easy to understand and manipulate. For this π-ADL operates on the principle of first class citizenship for all language elements. In light of this approach, there are specific syntactic features of the language that enable the manipulation and use of program elements with a minimal amount of code, with a syntax that is easy to understand.

The π-ADL.NET project is a compiler and runtime system development effort with the goal of compiling π-ADL to the .NET platform and running architecture specifications on that platform. The motivation for this project is to provide a large experimental space for testing and evaluating the process oriented, formally founded π-ADL in the context of a widely used software development and execution platform. At the same time there is a need to specialize the π-ADL language in order to leverage the functionality available in the reusable .NET software libraries.

This paper proposes language extensions to π-ADL in order for it to interface with .NET software libraries. The .NET platform is fundamentally object-oriented, and the need to represent .NET semantics while remaining within the confines of the process-oriented paradigm of π-ADL poses a modelling problem. Section II presents some general details of π-ADL in order to put the language extensions in context. Section III provides syntactic details of the proposed language extensions. Section IV presents related work with a comparison to the .NET API access features in other .NET languages based on formal methods. Section V concludes the paper.

II. π-ADL

π-ADL is an Architecture Description Language presented in [2] [4]. The purpose of Architecture Description Languages is to model software architectures, and according to [5] they focus on the high-level structure of the overall application rather than the implementation details of any specific source module. However π-ADL also provides programmatic constructs that allow it to model implementation details, so it can be seen as both a modelling and an implementation language.

π-ADL can be considered as a benchmark for second generation ADLs in the sense that it is formally founded and allows the customization of run-time architectural concepts. These two features were not present in earlier formal ADLs.

* This work is supported by the Higher Education Commission of Pakistan under the Program Overseas 2005-776582.

```
behaviour {
  x : Connection [Boolean];
  compose {
    via a1 send Void where {x renames b};
  and
    via a2 send Void where {x renames b};
  and
    via a3 send Void where {x renames b};
  }
}

value a1 is abstraction() {
       b : Connection [Boolean];
       via out send "Hello";
       via b send true;
}
```

```
value a2 is abstraction() {
       bb : Boolean;
       b : Connection [Boolean];
       via b receive bb;
       via out send " World";
       via b send false;
}

value a3 is abstraction() {
       bb : Boolean;
       b : Connection [Boolean];
       via b receive bb;
       if (bb == true) do {
              via b send bb;
              via b receive bb;
       }
       via out send "!\n";
}
```

Listing 1. A Simple Multi-Agent System in π-ADL

The second feature implies that the semantic definition of high level architectural elements can be customized, allowing a large amount of flexibility when defining architectural styles [6].

π-ADL is formally founded on the higher-order typed π-calculus (hence the name), which encompasses a formal transition and type system. It conforms to the language design principles of correspondence, abstraction and data type completeness [2]. Type completeness assures first class citizenship to all types i.e. they can be declared, assigned, can have equality defined over them, and can be persisted. The structural operational semantics of π-ADL represent behaviour (and thereby computation) by means of a deductive system, expressed by a formal transition system in line with the language type system. Type soundness asserts that well-typed terms do not give rise to runtime errors under the transition system.

π-ADL has a layered syntactic structure, with a core language layer forming the foundation of higher layers that build upon one another. For example high level architectural concepts such as components and connectors are expressed in terms of behaviours and abstractions, which are fundamental units of execution. A behaviour is an independent unit of execution launched when a process starts executing. Abstractions are abstractions over behaviours as functions are abstractions over data. Abstractions need to be called from behaviours or other abstractions in order to issuing behaviours. In order to enable communication between different parts of a unit of execution, or amongst different units of executions, connections are employed. Send and receive statements allow synchronous communication through connections. π-ADL provides concurrency constructs in line with those of π-calculus, in the form of *compose*, *choose* and *replicate*. Sub-blocks inside a compose block will execute in parallel with each other, while from those inside a choose block, only one will execute. The replicate construct connotes the infinite replication of an execution block.

Arithmetic and assignment syntax in π-ADL is based on the familiar style of Java and C#. π-ADL supports a set of basic data types in the form of *Integer*, *Boolean*, *String* and *Float*; and a diverse set of constructed data types: *any*, *view*, *tuple*, *union*, *quote*, *variant*, *set*, *bag*, and *sequence* types. Listing 1 presents a sample π-ADL code highlighting the main syntactic features of the language. It demonstrates the modelling of a simple synchronized multi-agent system in π-ADL. Three parallel pseudo-applications of the abstractions a1, a2 and a3 are performed in the behaviour. The three abstractions then synchronize with one another to print the string "Hello World!" in the right order.

III. .NET EXTENSIONS TO π-ADL

We now present the proposed syntax extensions to π-ADL to provide support for accessing the .NET API. All details are presented from the π-ADL perspective, using elements defined in π-ADL to model the extensions. An understanding of the .NET language framework is assumed.

The focus of this current set of extensions is to allow the use of existing .NET libraries in π-ADL.NET, and not develop reusable components in π-ADL.NET for access through other .NET languages. The latter could form the focus of later work on π-ADL syntax extensions.

A. Declaring and Instantiating Classes

Let NET be the .NET governor behaviour, in terms of the π-ADL formalism. We declare that NET has connections type_in and type_out, and they both process values of type *Any* (i.e. values of any type). NET supports a typing system expressed by the .NET namespace value attached to each type it receives or sends e.g. "System.String". The π-

ADL type system is therefore extended to be able to evaluate the .NET namespace notation and hence recognize the .NET types. All .NET objects are treated as abstractions in π-ADL.

The instantiation of a .NET class would be as follows:

```
x:System::Text::StringBuilder; //declaration
via NET::type_in send
"System::Text::StringBuilder";
via NET::type_out receive x;
```

Here the type_in connection receives a string value indicating the type of object it will process. The NET behaviour then internally creates the object and sends it out via the type_out connection. A shorthand notation forming the language syntax for this would be an overload of the '=' sign connoting instantiation, with the restriction of using parenthesis after the class name:

```
x:System::Text::StringBuilder; //declaration
//instantiation:
x = System::Text::StringBuilder();
```

x would now contain a reference to an abstraction of type System::Text::StringBuilder.

B. Namespaces

In order to recognize the .NET namespaces and incorporate them for usage in π-ADL programs, we specify the namespaces being used in a π-ADL program using the *use* directive. It is used at the program level, outside behaviour or abstraction definitions. Therefore:

```
//... declaration(s)
x = System::Text::StringBuilder();
```

can be rewritten as:

```
use System::Text;
//...behaviour header
//... declaration(s)
x = StringBuilder();
```

Note that the namespace System is implicitly used and need not be declared using the use keyword.

C. Constructors

The statement

```
x = System::Text::StringBuilder();
```

would result in NET internally calling the default constructor of StringBuilder. In case a constructor with arguments needs to be called, we can employee NET::type_in as follows:

```
//invoking the StringBuilder constructor
// with an Integer argument defining
//initial length
//... declaration(s)
via NET::type_in send
      "System::Text::StringBuilder";
via NET:: type_in send 42;
via NET:: type_out receive x;
```

Generally, the arguments to the constructor will be sent via type_in to the NET behaviour, right after sending the string that contains the type of object that needs to be created. The shorthand notation to this syntax would be:

```
x = System::Text::StringBuilder(42);
```

Multiple arguments will be sent as a comma separated list e.g.

```
y = MyObject("arg1", true, 0.5);
```

D. Methods, Fields and Properties

In order to incorporate the use of methods in π-ADL, we develop an internal model of a .NET class that is recognizable in the π-ADL syntactic domain, and covers interaction with both static and non-static fields and methods. Each .NET class is represented by a corresponding behaviour, and serves to provide access to static fields and methods. Conforming with π-ADL syntax this behaviour contains publicly accessible variables (for representing static fields).

Static methods are treated as connections. Each connection receives an argument of type any. It also makes a return value available via an input prefix (for methods that have non-void return values). In order to model non-static fields and methods, a similar approach is used with the difference that abstractions and not behaviours represent objects.

Listing 2 shows a .NET class and Listing 3 shows how it would look like in π-ADL. Notice that a .NET class is fully expressed by a behaviour and an abstraction, to model the static and non-static aspects of the class respectively. The following example shows how fields and methods for .NET objects are accessed in π-ADL:

```
x : PiADL::Vector; y : Float;
x = PiADL::Vector(3,4);
compose {
      via x::Resultant send Void;
and
      via x::Resultant receive y;
}
```

The shorthand equivalent of the above code would be

```
x : PiADL::Vector; y : Float;

x = PiADL::Vector(3,4);
//shorthand for the method call compose
y = x::Resultant();
```

```
public class PiADL.Vector {
  private double _x, _y;
  public double X {
    get { return _x; }
    set { _x = value; }
  }
  public double Y {
    get { return _y; }
    set { _y = value; }
  }
  public Vector(int x, int y) {
    _x = x;  _y = y;
  }

  public static string Name()
  {
    return "Vector";
  }
```

```
public static string Space()
  {
    return "PiADL";
  }

  public double Resultant()
  {
    return Math.Sqrt(_x*_y);
  }

  public double Angle()
  {
    return Math.ATan(_y/_x);
  }
} //end class
```

Listing 2. .NET class (using C# code)

```
Vector names behaviour
{
  Name : Connection[any];
  Space : Connection[any];

  compose
  {
    replicate {
      via Name receive;
      via Name send "Vector";
    }
  and
    replicate {
      via Space receive;
      via Space send "PiADL";
    }
  }
} //end behaviour

value VectorInstance is abstraction {
  X : Float; Y : Float;
```

```
Resultant : Connection[any];
Angle : Connection[any];
dVal : Float;

compose
{
    replicate {
      via Resultant receive;
      dVal = X * Y;
      dVal = Math.Sqrt(dVal);
      via Resultant send dVal;
    }
  and
    replicate {
      via Angle receive;
      dVal = Y / X;
      dVal = Math.ATan(dVal);
      via Angle send dVal;
    }
} //end abstraction
```

Listing 3. π-ADL model for the .NET class

Fields and properties are treated as shorthand projections e.g.

```
x::Length = 42;
```

There is also a case of indexed properties, which are treated from the programmer's perspective as an array. Both arrays and indexed properties are accessed using square brackets enclosing the index value, such as:

```
a : System::Collections::ArrayList;
o : System::Object;
//initialize and populate a
o = a::Item[4];
```

E. Events

Abstractions handle events generated by .NET objects. The handles keyword will allow the assignment of an abstraction to handle a certain event type. The proposed syntax is:

```
use System::Windows::Forms;
behaviour {
       t : TextBox;
       t = TextBox();
       t::Click = a;
}
value a is abstraction (x : Integer)
       handles TextBox::Click {
//....
}
```

F. Enumerations

A .NET enumeration is modelled as a π-ADL view type, with all elements of type Integer.

G. Exceptions

Since exceptions are full-fledged classes in .NET, they can be treated using the same behaviour-abstraction model described in III.A. above. Examining the exception properties and variables, and calling its methods is the same as described in III.A. The exception handling syntax is the try-catch-finally approach seen in C#, as shown below:

```
s : String;
i : Integer;
e1 : FormatException;
e2 : OverflowException;
try {
       via out send "Enter an integer: ";
       via in receive s;
       i = Convert::ToInt64(s);
       i = i * i;
       via out send i;
}
catch (e1, e2)
{ via out send "Invalid input value."; }
finally {
  via out send
    "\n\n End of try-catch-finally demo.\n";
}
```

H. Null Values

.NET objects initialized as Null values can be recognized and compared using the null keyword. All .NET objects are assumed null at the time of declaration.

I. Delegates

.NET delegates are represented using the connection renaming syntax in π-ADL. For example we apply this proposed syntax for delegates to the class definition in Listing 3:

```
m : Connection[Any];
m renames Vector::Name;
```

J. Casting to and from System::Object

System::Object is the canonical root class from which all .NET classes are derived. Consequently, .NET collection types often cast into System::Object the elements they are collecting. In the π-ADL.NET compiler, each of the basic types corresponds to one of the primitive types of the .NET platform. The .NET platform provides the ability to box primitive types into representative objects e.g. the System::Int64 structure exists for 64-bit integers and so on. In π-ADL.NET the constructed types are implemented as .NET classes and hence are directly cast-able to and from System::Object. Therefore the following proposed syntax for casting π-ADL types to and from objects is easy to implement:

```
c : System::Collections::ArrayList;
v : view[a : String, b : Boolean];
o : System::Object;
i : Integer;

c =  System::Collections::ArrayList();
i = 5;
v = view(a : "Cast test", b : true);

c::Add(i); //implicit cast to System::Object
o = (System::Object)v; //explicit cast
c::Add(o);
v = (view)c[1]; //explicit cast
```

K. Upcasts and Downcasts

Following the syntactic convention in III.J, .NET objects can be cast to any one of their inherited types and back. As such casts are dynamically checked in .NET, any runtime errors resulting from incorrect casts will have to be handled using the exception handling mechanism discussed in III.G. The following example illustrates the syntax:

```
use System::Windows::Forms;
behaviour {
       b : Button; c : Control; o : Object;
       b = Button();
       c = (Control)b; //upcast
       o = (Object)c; //upcast
       c = (Control)o; //downcast
       b = (Button)c; //downcast
}
```

L. Generics

Since our current focus in .NET extensions for π-ADL.NET is to provide the ability to use existing class libraries only, and not to be able to create new ones, we need not go into a detailed syntax mapping for .NET generics. It is sufficient to be able to declare and instantiate a .NET generic class as follows:

```
c : GenericClass<Integer>; //declaration
c = GenericClass<Integer>(); //instantiation
```

For generic class with multiple generic parameters, the parameters can be comma-separated. From the π-ADL perspective, the class GenericClass<T> represents a family of behaviour-abstraction pairs, each one of which processes a certain data type. Note that this model is compatible with constrained generic classes as well [12]. π-ADL.NET will also be able to use its own basic and constructed types as type parameters when instantiating a generic.

M. Inner classes

Just as a regular .NET class is fully defined by its name and namespace, an inner class can be defined by its name, container class and namespace. From the π-ADL.NET perspective a public inner class is treated in the same manner as an ordinary class. For example if a class Container contains the class Inner, and is declared in the namespace MyNameSpace then the following π-ADL.NET code will be valid:

```
i : MyNameSpace::Container::Inner;
i =  MyNameSpace::Container::Inner();
```

The π-ADL.NET code then can access the members of Inner like for any other class or object.

IV. RELATED WORK

The work presented here merits a comparison with a similar work performed for other formally founded languages. One such language is F# [7], a .NET language based on ML [8]. Both languages are based on formal methods, although F# is a functional language while π-ADL is a process oriented language. But in each case, the foundations of these languages differ greatly from that of .NET, which is principally object oriented. Furthermore F# provides syntax level interoperability for all the .NET features discussed in section 3 except for generics, thus providing an almost equivalent level of access to the .NET API, when compared to the π-ADL .NET extensions. However F# has been designed as a .NET language right from the start, whereas π-ADL as a language is neutral to any platform technologies.

Another ML based language for the .NET platform is SML.NET [11] based on Standard ML '97. It supports most of the features discussed in section III, but does not provide support for events or generics.

L Sharp.NET [9] is an implementation of the Lisp functional programming language for the .NET platform. It provides support for namespaces, object instantiation, and access to static and non-static methods, fields and properties. Advanced features such as casting, event handling, exceptions and generics are however not supported. DotLisp [10] is a lisp like interpreted .NET language with limited support for .NET types and delegates, as well as exception handling. It does not allow the instantiation of objects or the use of namespaces.

In short when it comes to compilers and extensions to formally founded languages for .NET, the current body of related work restricts itself to functional languages. Plus as seen in this section, none of these languages provide support for generics, an important feature of .NET version 2.0 [3]. Seeing this reported work from another angle, there is no Architecture Description Language compiler for .NET besides π-ADL.NET. This work is thus a significant overture in that it provides the possibility of examining a process-oriented, formally founded Architecture Description Language in the context of the .NET universe.

V. CONCLUSION

The purpose of the .NET extensions to π-ADL is to provide syntactic basis for the π-ADL.NET compiler to interact with .NET libraries. At the same time, these extensions form a reference for interfacing π-ADL with other software technologies. The development of these extensions is particularly important in that the architecture oriented focus of π-ADL lends naturally to the notion of employing ready-made software components to put together a software system, or perhaps roll out a software development project in which both architecture and components evolve through interaction with each other.

One goal of the π-ADL.NET project is to allow software architectural modelling on a mainstream software development platform. The implementation of these proposed extensions will enable the development of software using multiple paradigms and languages simultaneously. This helps meet the universal goals of reliable and low-cost software development through the following advantages: effort put in defining the software architecture is employed directly in the resulting software system; the privilege of an architectural view of the system that can be analyzed through compilation and execution; and bringing the benefits of the architectural paradigm to the rich technological foundations of .NET.

REFERENCES

[1] Medvidovic N., Taylor R., "A Classification and Comparison Framework for Software Architecture Description Languages", IEEE Transactions on Software Engineering, v.26 n.1, p.70-93, January 2000.

[2] Oquendo F., "π-ADL: An Architecture Description Language based on the Higher Order Typed π-Calculus for Specifying Dynamic and Mobile Software Architectures," ACM Software Engineering Notes, No. 3, May 2004.

[3] Lowy J., "Programming .NET Components 2nd Edition Appendix D: Generics," ISBN: 978-0596102074, O'Reilly Media Inc., 2005.

[4] Oquendo F., "Tutorial on ArchWare ADL – Version 2 (π-ADL Tutorial)," ArchWare European RTD Project IST-2001-32360, Jun 2005.

[5] Vestal S., "A Cursory Overview and Comparison of Four Architecture Description Languages," technical report, Honeywell Technology Center, 1993.

[6] Oquendo F. et al, "ArchWare: Architecting Evolvable Software," Proceedings of the 1st European Workshop on Software Architecture, LNCS 3047, Springer Verlag, May 2004.

[7] Syme D. and Margetson J., The F# website, 2007. See http://research.microsoft.com/fsharp/.

[8] Gordon M., et al. "Edinburgh LCF: A mechanised logic of computation," Lecture Notes in Computer Science, volume 78. Springer Verlag, 1979.

[9] Blackwell R., The L Sharp.NET website, 2007. See http://www.lsharp.org/.

[10] Hickey R., The DotLisp website, 2007. See http://dotlisp.sourceforge.net/dotlisp.htm.

[11] Benton N. et al. "Adventures in interoperability: the SML.NET experience," Proceedings of the 6th ACM SIGPLAN International conference on Principles and Practice of Declarative Programming, ACM Press, 2004.

Towards Collaborative Development Based on Software Architecture

Yanchun Sun, Hui Song, Xinghua Wang, Wenpin Jiao
Institute of Software, School of Electronics Engineering & Computer Science, Peking University,
Key laboratory of High Confidence Software Technologies, Ministry of Education,
Beijing 100871, P.R.China
E-mail: {sunyc, songhui06, wangxh05, jwp}@sei.pku.edu.cn

Abstract

Software projects often require many software engineers to coordinate their efforts to build large software systems. How to support collaborative development among the stakeholders in a project, even if separated by time or space, to produce software artifacts efficiently becomes a very important problem. Software architectures are considered important because they are the blueprints for target software products and determine system-wide qualities, and they can be used to organize various software artifacts in software development process from a high level perspective. Based on such an important role of software architectures, this paper puts forward an approach to collaborative software development based on software architecture, to support the collaboration spanning the whole software lifecycle. Moreover, the paper provides the detail on how this method works efficiently by a case study.

1. Introduction

Software engineering projects are inherently cooperative, requiring many software engineers to coordinate their efforts to produce a large software system [1]. How to support collaborative development among the stakeholders in a project to produce software artifacts efficiently becomes a very important problem.

Typical software development methods provide collaboration support from two sides: one is to provide various communication mechanisms between software developers; the other is to provide collaborative support based software artifacts for software developers. For the first category, there are many kinds of communication mechanisms such as email, instant message, video meeting, etc. However, these communication mechanisms are based on natural language and quite short of standardization, which is easy to lead to ambiguous understanding for developers. On the other side, collaborative development method based on software artifacts can solve this problem if software artifacts have good structure, clear syntax and explicit semantics.

For the collaboration based on software artifacts, the version control systems are often used to manage the software artifacts, for instance, CVS, SVN, etc. These tools are very important for collaborative development among software engineers, but the artifacts stored in these tools are almost code-level programs, and lack a reasonable organization from the viewpoint of software development process. As a result, these tools are short of support for collaborative design and maintenance of software, which usually require the designers or maintainers to have a big picture about the system.

To support the collaborative development in the whole software lifecycle, it is necessary to provide software developers with an appropriate model to organize various software artifacts in software development process from a high level perspective. Then, software engineers can collaboratively develop software based on this model.

With software becoming large and complex, software architecture (SA) becomes a blueprint to guide the development and maintenance of software systems, and it plays an extremely important role in the whole software lifecycle. Now, some SA-centered development methods have been put forward. Richard Taylor and David Garland present their own Architecture Description language (ADL) and propose the SA-centered development method based on the ADL [2,3]. Siemens's Hofmeister etc. describe a set of architecture views and put forward a corresponding software development method from requirement to implementation [4]. IBM's tool "RSA" (Rational Software Architecture) also focuses on SA-centered development [20]. In a conclusion, SA has become the core of software artifacts in software development process and an ideal base for collaborative development. Now many SA-centered software development methods and tools (e.g., ArchStudio[5] and MolhadoArch[6]) have partly supported the collaboration among developers. But these tools do not use the semantics of SA adequately, and they just support simple collaboration for several authors based on the management of authority.

In this paper, we put forward a collaborative development approach based on SA. First, based on version control tool and semantic information of SA, we abstract

the information of fine-grained modification into SA in order to support designers to collaboratively design SA model. Then we can enlarge this collaboration mechanism from collaborative design to the whole lifecycle because SA is a core artifact in the whole software lifecycle. Via introducing bi-transformation technologies [7], we can use architectural knowledge based on version control tool to capture the transformation relationship between SA and other artifact to transform the modification manipulation of other artifacts to the modification manipulation of SA model, so as to support the collaborative development among different developers.

The rest of this paper is organized as follows. Section 2 presents some related work. In Section 3, we put forward an approach to collaborative development based on software architecture. Section 4 illustrates the approach with a case study. Section 5 concludes the contribution of the paper and gives the future work.

2. Related work

To the best of our knowledge, the researches on collaborative development based on SA cover several aspects. We organize the review of the related work from five main aspects as follows.

(1) The collaboration support for different development phase. Many SA development tools support collaborative authoring by versioning architecture description files, e.g., ArchStudio and ACMEStudio [8], but this kind of support for collaboration is fine-grained without the semantic information of SA. Thus, the support is just limited in the phase of SA design rather than the entire software lifecycle.

(2) **The support for communication mechanism of collaboration.** Our work is different from some researches which just use the mechanism of communication to support collaboration [9]. Our work supports the collaboration based on models, and it can eliminate the ambiguity induced by the communication in natural languages. On the other hand, these mechanisms of communication are also a supplement to our work and easy to be integrated to our framework.

(3) **The synchronization of artifacts in collaboration.** Our collaborative mechanism inherently supports the synchronization of artifacts, in other words, the synchronization among several replications of an artifact. Compared with the research on optimistic replication [11], our work can support large-granularity synchronization and even the synchronization of heterogeneous replications.

(4) **The visualization of collaboration.** One key issue in collaboration is how to assist the collaborative developers to know other person's work status (i.e., awareness) [10]. There are some related researches, in which the mechanism of these researches is similar to that of our work. That is, based on version control tool, to get the low-level information of modification and display the information in a direct way [12,13]. But their work is to show different detailed information from several views, and we organize the detailed information of modification in multi-view of software architecture. Another difference is that this visualized view is the threshold of collaborative development, and not just a static visualized display.

(5) **Integrated collaborative development tool**. There are some researches focusing on the integration of several communication tools [14,15,16]. IBM and Microsoft both have their commercial products to support collaboration. The core of these products is to abstract the common mechanisms from typical collaboration and communication, and integrate them together. However, our work is from the view of the development method based on SA, and discusses potential issues for collaboration. Moreover, most of these tools support collaborative communication based on fine-grained code, but we try to organize the collaborative development from a high level at SA.

(6) **Web based collaborative development environments.** Eclipse is an open source community. Its project focuses on providing a set of development platform and framework to facilitate the construction of software. Every eclipse project has an independent page as the entry to all tools based on the Internet, which includes identity authentication, CVS, mailing list, newsgroup, WiKi [17], and Bugzilla [18]. Now, our work focuses the integration surrounding IDE, and how to extend it to the Internet environment will be our future research work**.**

3. Our Approach

The core of our approach is maintaining the different versions of an architecture model, and revealing the change in the proper views of an architecture model, on the basis of the semantics of this architecture model. An overview of our approach is illustrated as Figure 1.

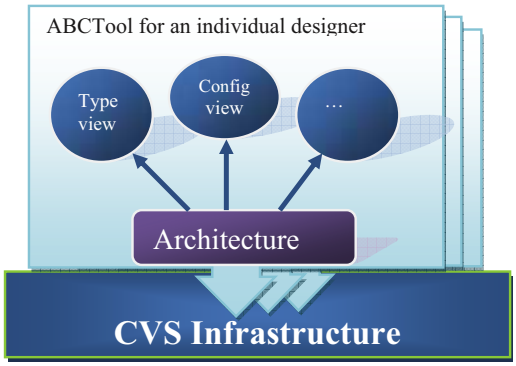


Figure 1. Approach Overview

In the rest of this section, we present several key aspects of our approach supporting collaborative development.

3.1. Software Architecture Model

By referencing the typical architecture description language (ADL) [19], we define the meta-model(described as Figure 2) of software architecture by using Eclipse Ecore. The core concept of this meta-model is *Component*. We partition and organize the software into components, each with a relatively individual concern. Components are associated with connections which link the provided and required interfaces together.

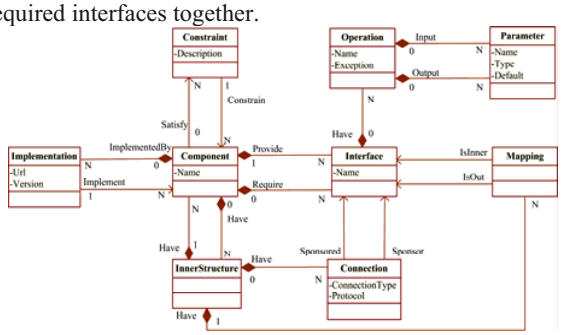


Figure 2. Software Architecture Model in Our Approach

We also introduce the concept of *InnerStructure,* which helps organizing the whole system as a hierarchical structure to support the stepwise refinement during architecture design. Based on the meta-model, we construct a software architecture modeling environment by using Eclipse GMF, named ABCTool, in order that we could assist designers to record their design decisions by recording their manipulations such as additions and deletions of elements and modifications of properties and relationships of elements.

The entire architecture model is recorded in the form of XMI in several files. Among the files, one set of files record the model information of the core elements, and another set of files record every view's diagram information, including diagram color, font and the layout of elements.

3.2. Version Control in Low level

Designers can modify the SA model via the graphic interfaces, including core model, elements and layout. When the designers finish and save the modification, some related XMI files will be changed. As pure context files, these XMI files can be managed by a version control tool. We select CVS to achieve this because ABCTool is based on Eclipse platform which provides fully support for CVS.

By using CVS, we can record who makes the modifications and what modifications have been made to SA model in the collaborative development process. We use Eclipse Plug-in to encapsulate the record file of these modifications and visualize them in ABCTool.

3.3. Visualization of the SA Model Modifications

By CVS interface, we can obtain the information about the modifications from the XMI files. By analyzing the modifications information, we can elicit which elements in SA model have been modified and what kind of modifications has been made. Moreover, we can display the modifications explicitly in ABCTool, for example, using different color to distinguish added components, deleted components and unchanged components.

3.4. Model Maintenance by Collaborative Developers

For changed model, collaborative developers can select accepting the modification, rejecting the modification or adding new modification to the modification. The maintenance activities for the modification can be mapped to the operations in CVS. During the maintenance, collaborative developers can use the modification information offered by CVS to identify the intention of the modification. Sometimes, they may need to contact the modificator directly to discuss the goal of the modification. We provide a support mechanism for peer-to-peer communication in ABCTool.

3.5. The Collaborations among the Developers in Different Phases

As a core artifact in the entire software development process, SA model is a suitable medium for various developers from different phases to communicate. In different phases, developers will deliver different artifacts, but most of these artifacts record some core information of SA. In other words, some transformation relationships exist between these artifacts and SA model. Thus, by transforming the core information in SA and adding special information in a given phase, the artifact in the given phase can be constructed. Using those research fruits in the bi-transformation field [7], we can use a set of transformation rules to reflect the modification of SA model to other model, and also reflect the modification of SA level information in other model to SA model. Thus, we can utilize the approach above to assist the collaborations among the developers participating in different phases.

4. Case study

The following case illustrates how ABCTool assists different developers to collaboratively design a large software system. In the entire software development process, there are various artifacts, and these artifacts have different versions. Eclipse has integrated CVS to support collaborative development, but CVS just manages artifacts as context. For example, in Eclipse, we can see the history page of every file, including the version number, modification time, and author etc. According to different version, we can use command "Compare" to view the difference between two versions. In design phase, as core artifact, SA model is often composed of Component Type View and Component Configuration View etc. Thus, the management mechanism based on context in CVS is not enough. To solve this problem, our modeling tool "ABCTool" provides the visualized version management for the elements in Component Type View and Component Configuration View etc.

Furthermore, the current version control tools do not support synchronized update of artifacts, and developers have to initiatively update the artifacts to get the new version. So it is quite short of support for a group of people to collaborate their development, which often leads to low efficient collaboration. Aiming at this problem, ABCTool notifies the update to other collaborative developers when a developer updates an artifact. ABCTool will also display different updates for collaborative developers, for example, red color represents the update of "delete", green color represents the update of "add", and yellow color represents the update of "modify".

We will illustrate our approach described in section 3 by the following case. The scenario is as follows. Two developers collaboratively design software system "Example". One's user name is "cvsuer", and the other's user name is "wangxh". Configuration View "My.adl_cofig" (Shown as Figure 3) is one of main design artifacts of SA. It is stored in the server of CVS. We use file "My.adl.cofig" to describe how ABCTool assists the collaborative design among developers. Suppose that both developers are collaboratively developing the software based on version 1.2 in the design phase.

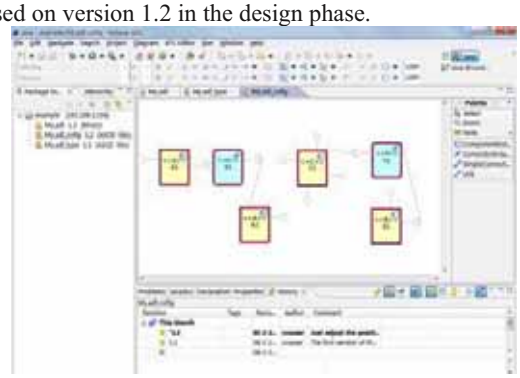


Figure 3. Component Configuration View "My.adl_cofig"

(1) User "wangxh" is editing file "My.adl.cofig" in the client. From the Page "History", we can see that there are two versions of this file, Version 1.1 and Version 1.2. In the server of CVS, newest version is 1.2. When user "wangxh" adds connector "Z1" and saves the file, we can see that the newest version in the client of User "wangxh" is still version 1.2 from the Page of "History" (described as Figure 4). Now the newest version is version 1.2 in the server. In the left navigator column, label ">" represents that the version of this file in the client is not consistent with the version in the server of CVS.

(2) After user ""wangxh" delivers the new version to the server of CVS, wangxh's client interface will be changed to Figure 5. We can see that the page of "History" shows the newest version in the client is version 1.3. We also see that file "My.adl_cofig" in the client is consistent with the file in the server from the left navigator column.

(3) When user "wangxh" submits new version, the page of "History" of another user "cvsuser" will display the version update of this file. At the same time, the tool will pop up a dialog box to remind user "cvsuser" that there is a new version has been submitted (described as Figure 6).

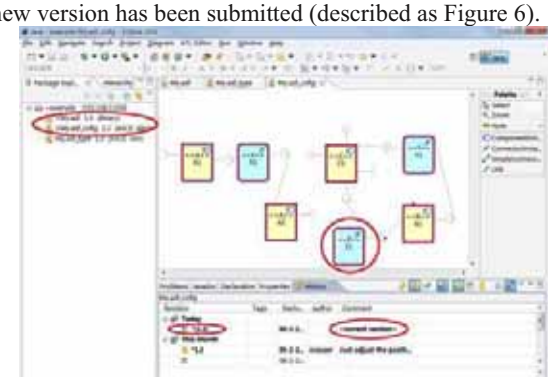


Figure 4. Wangxh' Client Interface before Version Update

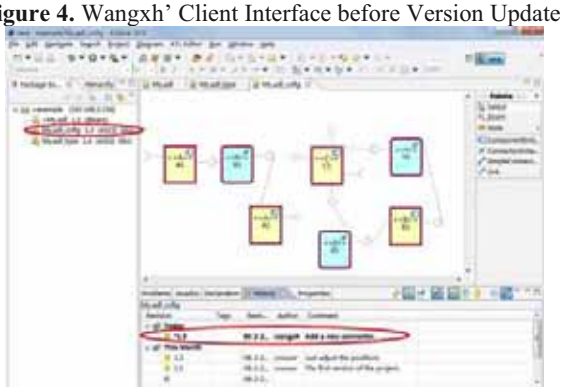


Figure 5. Wangxh' Client Interface after Version Update

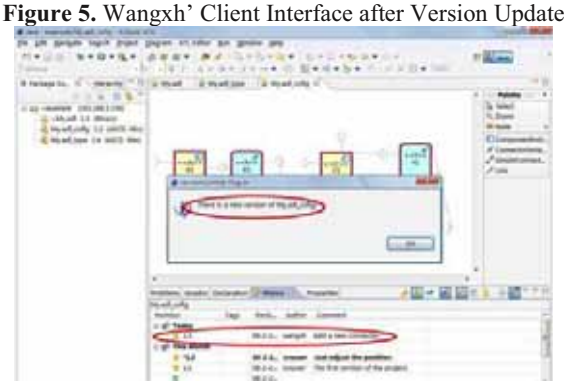


Figure 6. The Dialog Box to Remind the New Version

(4) When user "cvsuser" knows that the newest version of editing file "My.adl_cofig" is version 1.3, and its current version is 1.2, he can compare the difference between these two versions by the operation of "compare in ABCTool" (described as Figure 7). ABCTool can distinguish the classification of updates from their color. So, the new added connector will be highlighted green to represent its update is "add". Similarly, the element is yellow or red to represent the update is "modify" or "delete" respectively. The comparison based on the elements in diagram includes some kind of semantic comparison between these elements. So, it totally differs from the comparison based on text file.

We can see that the collaborative development supported by ABCTool is based on SA, and intuitionistic.

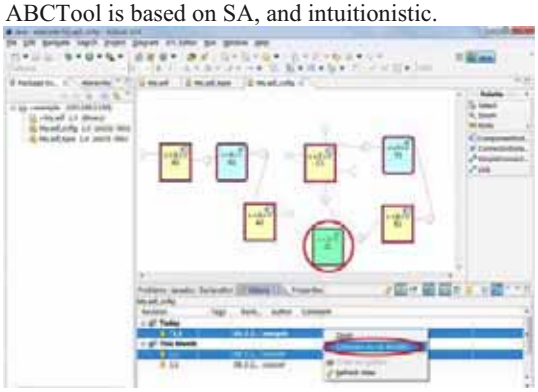


Figure 7. Compare the Difference between Two Versions

5. Conclusions

This paper puts forward an approach to support collaborative development in different phases based on software architecture. By using CVS, we can use software architecture model to distill the semantics of context changes recorded in CVS, and then use visualized SA modeling tool to display the change of SA model to assist different designers to collaborate their design. By using bi-transformation technologies, we can use the transformation relationships between SA and other artifacts to transform the modification manipulation of other artifacts from/to the modification manipulation of SA model and then use SA modeling tool "ABCTool" to support the collaborative development for different developers.

In the future, we will make further research on two key issues. One is how to provide more detailed controls on changes, including providing more semantics of change, introducing various developer roles, and managing authority. Another is how to introduce more architectural knowledge (e.g., design rationale) to facilitate the collaborative development.

Acknowledgement

This effort is sponsored by the National Basic Research Program of China (973) under Grant No. 2005CB321805 and the National Natural Science Foundation of China under Grant No.90612011, 60503028, 60773151 and the National High-Tech Research and Development Program (863) of China under Grant No. 2007AA01Z127, 2007AA010301.

References

[1] Jim Whitehead, "Collaboration in Software Engineering: A Roadmap", in *Future of Software Engineering(FOSE'07)*, Minneapolis, MN from May 19 to May 27, 2007.

[2] Nenad Medvidovic, David S. Rosenblum, Richard N. Taylor, "A language and environment for architecture-based software development and evolution", in *Proceedings of the 21st international conference on Software engineering*, Los Angeles, California, United States, 1999.

[3] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, Peter Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure"，*Computer*, vol. 37, no. 10, pp. 46-54, Oct., 2004.

[4] C Hofmeister, R Nord, D Soni, Applied Software Architecture, Addison Wesley, 2000.

[5] UCI Software Architecture Development Environment, 2007, http://www.isr.uci.edu/projects/archstudio.

[6] T.N.Nguyen and E.V.Munson, Object-oriented Configuration Management Technology can Improve Software Architectural Traceability", in 3rd *ACIS International Conference on Software Engineering Research, Management and Applications(SERA'05)*, Mount Pleasant, Michigan, USA, 2005, pp.86-93.

[7] Yingfei Xiong, Dongxi Liu, Zhenjiang Hu, Haiyan Zhao, Masato Takeichi, Hong Mei, "Towards Automatic Model Synchronization from Model Transformations", in *Proceedings of 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*, Atlanta, Georgia, November 5-9, 2007.

[8] A. Kompanek, "Modeling a System with Acme", 1998, http://www.cs.cmu.edu/~acme/html/WORKING-%20Modeling%20a%20System%20with%20Acme.html.

[9] Erran Carmel, Ritu Agarwal, "Tactical Approaches for Alleviating Distance in Global Software Development", *IEEE Software*, March/April, 2001, pp22-29.

[10] C. Gutwin, R. Penner, and K. Schneider, "Group Awareness in Distributed Software Development," in *Computer Supported Cooperative Work*, Chicago, Illinois, USA.: ACM Press, 2004, pp. 72-81.

[11] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Computing Surveys (CSUR),* vol. 37, pp. 42-81, 2005.

[12] S. G. Eick, J. L. Steffen, and E. E. Sumner Jr, "Seesoft-A Tool for Visualizing Line Oriented Software Statistics," *IEEE Transactions on Software Engineering,* vol. 18, pp. 957-968, 1992.

[13] S. G. Eick, T. L. Graves, A. F. Karr, A. Mockus, and P. Schuster, "Visualizing software changes," *IEEE Transactions on Software Engineering*, vol. 28, pp. 396-412, 2002.

[14] C. Cook and N. Churcher, "Modelling and measuring Collaborative Software Engineering," in *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, 2005, pp. 267-276.

[15] S. Dustdar and H. Gall, "Architectural concerns in distributed and mobile collaborative systems," in *Journal of Systems Architecture*. vol. 49, 2003, pp. 457-473.

[16] Y. Yang, "Separating data and control: support for adaptable consistency protocols in collaborative systems," in *Computer Supported Cooperate Work*: ACM Press New York, NY, USA, 2004, pp. 11-20.

[17] P. Louridas, "Using wikis in software development", *IEEE Software*, vol. 23, 2006, pp. 88-91.

[18] Mozilla, "Bugzilla," http://www.bugzilla.org/.

[19] N. Medvidovic and R. N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages", *IEEE Transactions on Software Engineering,* pp. 70-93, 2000.

[20] IBM, "Rational Software Architect Overview," 2007, http://www-306.ibm.com/software/awdtools/architect/swarchitect/.

Choosing a Software Architecture: An Approach and a Case Study

C. Ghezzi, G. Tamburrelli
Politecnico di Milano – Dipartimento di Elettronica e Informazione, Deep-SE Group
Piazza L. da Vinci, 32 – 20133 Milano, Italy
(ghezzi — tamburrelli)@elet.polimi.it

Abstract

The design of complex software systems requires careful analysis of alternative architectures, which may affect different qualities of an application. This paper reports on a case study in network management. Network management systems are traditionally designed according to a client-server paradigm. This work extends previous research that explored alternative solutions based on mobile code. Competing solutions are evaluated through analytical modeling. We also built a prototype design workbench, which supports more detailed comparisons of competing architectures by rapidly developing mockup implementations.

1. Introduction

Software architects are often confronted with the choice among different architectural paradigms in the design and implementation of complex software applications. Selecting the most appropriate style for the specific problem to be solved is not an easy task, and many factors influence the choice. Each choice, in fact, can have different effects on different properties that the resulting system should ensure. For example, one solution might be more robust, but might have a higher response time for certain functions. In general, no solutions can be found that work best in all possible practical situations, even for specific application domains and execution environments. It is therefore important to develop an approach that may support software architect in decision making.

In this paper we contextualize this general problem for an important class of critical software applications, namely network management (NM). NM is crucial for modern telecommunication networks, composed of thousands of interconnected nodes, which must be configured and monitored to guarantee a globally correct and efficient behavior. The network comprises many different kinds of devices: routers, switches, signal regenerators, repeaters. Because of distribution and accessibility problems, on site configuration and maintenance of individual nodes is often impossible. Rather, it must be performed by a remote NM system, which is responsible for collecting data from network apparatuses, monitoring them, and possibly reacting to the critical situations to guarantee the required quality of service.

NM is the discipline that studies the procedures and techniques used to monitor and configure the nodes of a telecommunication network. Besides conventional bookkeeping of on-line data and normal maintenance operations, NM is responsible for managing exceptional situations that can suddenly happen (e.g. broken links, traffic peaks). For example, to face a sudden overload of a particular network link, a performance management functionality might reconfigure a node by redirecting user traffic.

This paper is not strictly on NM. Rather, NM provides an interesting and realistic case study, which highlights how an architecture-driven software design process may provide a rational support to the reasoning and decision making of the software developer. We developed the case study in collaboration with industry[1], in the context of a feasibility study for the implementation of a new generation of NM services.

The currently adopted solutions to NM, such as SNMP [6] or RMON [14], adhere to a strict client-server paradigm, where the management station is in charge of querying nodes, collecting from them run-time monitoring data, and issuing appropriate response commands. The network management protocol specifies the format of messages and the semantics of operations. Because operations are described at a very fine-graine level, any NM functionality requires a large number of messages to be exchanged, and this *micromanagement* causes serious efficiency drawbacks in terms of bandwidth consumption and computational burden at the management station. The links connecting the management station to the rest of the network easily become bottlenecks. Another source of congestion can arise when new software has to be delivered through the network. For example, to install, upgrade or modify the firmware code on a network

[1]We wish to thank Marco Mussini of Alcatel-Lucent, who challenged us to work on software architectures for NM, and Michele Panzeri who participated in the project.

element, a point-to-point connection must be activated with every node. The installation may become necessary as a response to detected congestion symptoms. Paradoxically, to react to congestion, the management station has to collect data from network nodes, thus further contributing to congestion. Access to the nodes involved in NM can thus become difficult or sometimes impossible. Network congestion generates response time degradation and hinders network scalability.

A software architecture based on mobile agents (MAs) [1, 4, 8, 9] appears to provide an appealing alternative paradigm for the design of NM systems. MAs can be defined as computational entities that act autonomously—proactively and reactively—and can relocate themselves on different nodes of a network. More precisely, we refer to MAs that fall under the so-called *weak mobility* paradigm. Weak mobility implies that an agent moves from one node and restarts its execution as a new process at the destination node. It can, however, carry state information as global data, as we will see later on in more detail[2].

Intuitively, an MA can be injected in the network and instructed to reach a set of controlled nodes to perform some NM functionality. At each node the agent may collect the necessary data locally, process them, and then eventually move back to a control station to report its findings. The ratio between the size of the data that must be collected locally and the data transported to the control station is called *semantic compression* performed by the agent. Intuitively agents look like an appealing solution whenever the network functionality that has to be performed fosters a high semantic compression. In fact, this would reduce network traffic over an equivalent solution based on the traditional CS paradigm. More generally, a NM architecture based on MAs looks appealing for the following reasons [4]:

- *Load balancing*: computation is distributed on many nodes.
- *Availability*: the control station may be temporarily down while the agents injected in the network locally monitor the nodes.
- *Asynchronous, autonomous, parallel interaction*. Mobile agents may work in parallel.
- *On-line extensibility*. On-the-fly installation (for versioning and patching) is easy to achieve.
- *Analysis precision*. The agent installed in a node can collect precise real-time measurements, independent of the network latencies that would be involved in a CS solution.
- *Network traffic reduction*. By migrating code near the data it processes, network traffic may be reduced ([11, 13]).

Main goal of our study was to precisely evaluate CS vs. MA architectures to try to minimize network traffic. The first approach we investigated was based on a simple mathematical model, which allows the designer to reason in a quantitative fashion about network traffic. This part of our contribution extends previous work [2]. Because of the limitations implicit in analytical reasoning, our next contribution has been directed to developing an environment that would support rapid prototyping. In this environment, it is possible to perform an empirical analysis of network traffic in the different cases.

Our experience confirms the need for the software engineer to be supported by a suitable methodology and tools while reasoning about different architectural alternatives. In fact, there is no winning solution in general, even in domain-specific cases, like NM. The architecture that performs better should be chosen depending on the specific functionality to provide. Also, the two approaches to reasoning—analytical modeling and prototyping—are complementary. In fact, the former may ignore some relevant phenomena that occur in reality and sometimes it requires input data that are difficult to anticipate. On the other hand, the latter may provide an estimate for such data to drive a more refined analytical modeling. Although our approaches to architectural reasoning are presented here to evaluate CS vs. MA, we argue that similar approaches can be followed in other cases of competing architectures, and can be extended to quality attributes other then network traffic.

This paper is organized as follows: Section 2 details the comparison of CS and MAs via analytical modeling. Section 3 describes the experimental workbench we developed for rapid prototyping. It also illustrates a simple case study adopted as proof of concept, which focuses on analysis of network traffic generated by a classification task performed by a neural network, implemented using both paradigms. Finally, Section 4 draws some conclusions, and discusses the general lessons learned from the case study.

2. Architectural Reasoning via Analytical Modeling

Previous work by Picco and Baldi [2] addressed this issue by comparing CS and MA on networks with uniform communication cost (e.g. a fully connected network). Their work describes a simple analytical model through which the alternative architectures are easily compared. Unfortunately, however, their work does not apply to several common practical cases. In particular, large telecommunication networks, for which NM is crucial, usually have a ring

[2] A taxonomy of mobile agents and the notion of weak mobility versus strong mobility have been introduced in [7].

topology. In this section we extend the approach of Picco and Baldi to networks with a ring topology, using the same formal notation. Let us introduce the following entities:

- η_{CS} = Protocol Overhead for Client Server
- η_{MA} = Protocol Overhead for Mobile Agents
- I = Request Size
- R = Reply Size

We assume the network to be composed of N nodes and, in a CS architecture, the management station performs Q requests of size I to each node, which replies with messages of size R. Instead in a MA architecture, an agent is injected in a node by the control station, with the purpose of letting it visit possibly all nodes of the network and performing some actions in each of them, consisting of issuing Q requests of size I and receiving replies of size R. After all nodes are visited, eventually the agent migrates from the last node back to the control station.

2.1. Total Network Traffic

The total traffic generated by a MA on a ring topology network can be modeled by the following formula:

$$T_{\text{MA}} = \sum_{n=1}^{N}(C_{\text{MA}} + S_{\text{MA,n}}) + \eta_{\text{MA}} \sum_{n=1}^{N}\sum_{q=1}^{Q} R_{q,n} \qquad (1)$$

The first term describes the traffic generated by an agent during its relocations: every agent has a size that is determined by a code fragment C_{MA} and a state $S_{\text{MA,n}}$. The second term describes the replies from the nodes, i.e., the data collected at each node, which must be communicated to the control as illustrated in Figure 1.

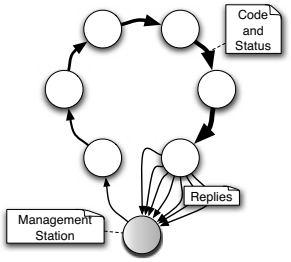


Figure 1. Mobile Agents communication flow

The state of the agent in each node is composed of the replies collected by the agent during its relocations. It is thus possible to write:

$$S_{\text{MA,n}} = \begin{cases} 0, & \text{if n=1} \\ \sum_{m=1}^{n-1}\sum_{q=1}^{Q} R_{q,m}, & \text{if n>1} \end{cases} \qquad (2)$$

Indeed for every node the status is the sum of the previous statuses (i.e. the information collected in the previous nodes). Assuming that the size of replies is a constant R we can write:

$$S_{\text{MA,n}} = \begin{cases} 0, & \text{if n=1} \\ \sum_{m=1}^{n-1} QR = (n-1)QR, & \text{if n>1} \end{cases} \qquad (3)$$

Consequently:

$$T_{\text{MA}} = \sum_{n=1}^{N} \eta_{\text{MA}}(C_{\text{MA}} + (n-1)QR) + \eta_{\text{MA}} NQR = \qquad (4)$$

$$= \eta_{\text{MA}}(NC_{\text{MA}} + \frac{N(N+1)}{2}QR)$$

Obviously, under the same assumptions made for MAs, for the CS paradigm the cost in a ring topology network is not uniform because the cost of communication depends on the distance in the ring between the management station and the node of interest.

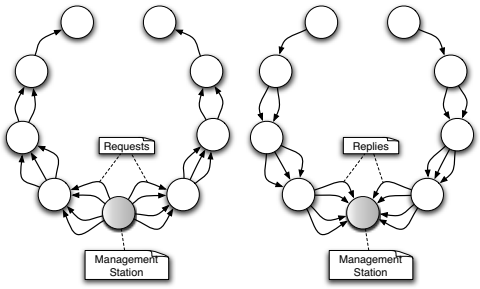


Figure 2. Client Server communication flow

Thus the total traffic can be modeled by the following weighted sum (see Figure 2):

$$T_{\text{CS}} = 2\sum_{m=1}^{\frac{N}{2}} mQ(\eta_{\text{CS}} I + \eta_{\text{CS}} R) \qquad (5)$$

where m represents a multiplicative factor (m = N/2 if we consider the traffic at the management station; m = (N/2)-1 at the adjacent nodes, and so on). The formula can be rewritten as:

$$T_{\text{CS}} = \frac{N}{2}\left(\frac{N}{2}+1\right) Q(\eta_{\text{CS}} I + \eta_{\text{CS}} R) \qquad (6)$$

At this stage, it is necessary to take into consideration the semantic compression. As we observed earlier, in fact, agents can compress the data collected. Thus the size of the reply computed by a MA at a given node can be a fraction of the size of the data that are transmitted back to the

management station in a CS setting. Assuming the size of the reply in the CS case to be a constant R and assuming the average reduction factor to be a constant θ, the size of the replies for a MA at each node is θR. T_{MA} then becomes:

$$T_{\text{MA}} = \eta_{\text{MA}}(NC_{\text{MA}} + \theta\frac{N(N+1)}{2}QR) \quad (7)$$

Moreover, assuming N to be large, and $\eta_{\text{MA}} \approx \eta_{\text{CS}}$, we obtain that $T_{\text{MA}}<T_{\text{CS}}$ if and only if:

$$C_{\text{MA}}<\frac{N}{4}Q(I+R) - \theta QR\frac{N}{2} \quad (8)$$

This inequality describes under which circumstances MAs are preferable to CS in terms of total traffic generated in a ring topology network. The MA code size is compared with a term that depends on number of nodes, number of queries, request and reply size and average semantic compression.

MAs can perform even better if *global semantic compression* can be achieved on replies. As an example, let us consider a scenario in which an agent travels through the entire network, computing at each node a compressed value, and delivering to the control station the maximum of such values. In this case the agent has to carry through the network only a single reply R containing the current maximum. Thus T_{MA} becomes:

$$T_{\text{MA}} = \eta_{\text{MA}}N(C_{\text{MA}} + R) \quad (9)$$

$T_{\text{MA}}<T_{\text{CS}}$ if and only if:

$$C_{\text{MA}}<\frac{N}{4}Q(I+R) - R \quad (10)$$

2.2. Network Traffic at the Control Station

Network management can generate intense traffic around the control station. Intuitively, this is especially true in CS cases. Thus, instead of analyzing the total network traffic, it may be interesting to evaluate the possible bottlenecks around the control station. The traffic generated by the CS paradigm is:

$$T_{\text{CS}} = \eta_{\text{CS}}NQ(I+R) \quad (11)$$

The formula represents the transmitted requests I and the received replies R between the control station and two adjacent nodes inside the ring (see Figure 2). On the other hand, for MAs we have:

$$T_{\text{MA}} = \eta_{\text{MA}}C_{\text{MA}} + \eta_{\text{MA}}NQR \quad (12)$$

Consequently $T_{\text{MA}}<T_{\text{CS}}$ if and only if:

$$C_{\text{MA}}<NQI \quad (13)$$

where NQI represents the total size of the requests. Moreover, considering semantic compression, we obtain:

$$C_{\text{MA}}<NQ(I+R) - \theta NQR \quad (14)$$

This inequality clearly states that MAs perform better if the agent's code is smaller with respect to the difference between the (semantically compressed) data collected by an agent and the size of the total requests and replies transmitted/received by the control station. Furthermore, if we analyze the traffic at the management station in case of global semantic compression we obtain:

$$C_{\text{MA}}<NQ(I+R) - R \quad (15)$$

We may conclude that the comparison of CS and MA architectures in terms of the generated traffic depends on several parameters of the application. Some of the data involved in the formulae may be difficult to anticipate, and might require some experiments to be performed in order to obtain them. We believe that in general this kind of analysis may benefit from a complementary kind of experimental analysis. The prototype workbench described in the next section has been developed to support this task.

3. Prototype Workbench

The workbench offers features that allow the designer to rapidly develop mockups of NM functionalities and run them to get comparative quantitative data (e.g., network traffic). Mockups comprise a set of sites, connected through a network. A middleware, called *playground*, supports communication and coordination of NM mockup components that run on top of it. Each site provides its own playground. To support rapid prototyping, we decided to use scripting languages to implement NM functionalities. Each functionality is implemented by what is called an *agent*. Agents should not be confused nor identified with MAs. They can be programmed to behave as MAs, but they can also support other paradigms, such as CS. Agents may be written in any scripting language. We developed support for Perl, Python, and Ruby, but others may be easily developed. A decoupling layer (i.e., the tool layer), provides an API that allows agents to transfer their execution from a playground to another and allows communication mapping the data provided by agents onto linear messages transferred by the middleware. Figure 3 shows a fragment of the architecture.

An agent defines an elementary functional unit. It is represented by a tuple <C, B, D>, where C is a code block that defines a computation, B is a dataset (called backpack) that stores useful information for the agent, and D is the agent's descriptor. D stores the static and dynamic description of an agent (interface, intention, version, author). If an agent implements an MA that collects data in each visited node, collected data are stored in the backpack. In such a case, the tuple that represents an agent is transferred from a playground to another as an XML file.

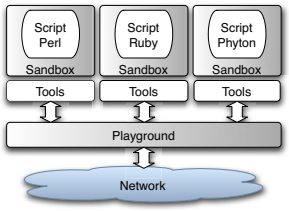


Figure 3. System Architecture

Agents may communicate via asynchronous messages. A message is defined by the pair <*E, Y*>, where E is the message envelope and Y is the body, which represents the information to transfer, and has variable length. E is also a pair <*S, R*>, where S identifies the sender and R identifies the receiver(s). S is defined by a pair <*IDsite, IDagent*>, which uniquely identifies an agent. Because the middleware supports both unicast and broadcast (at site level or at agent level), R can have one of the following structures:

- <*IDsite, IDagent*>: unicast.
- <*IDsite, **>: site unicast, agent broadcast.
- <*, *IDagent*>: site broadcast, agent unicast.
- <*, *>: site broadcast, agent broadcast.

An agent that wishes to send the message must provide information about the receiver(s) and the body. The communication tool automatically generates the rest of the message. Once sent, the message is passed to the middleware and then routed to destination. Once it arrives at destination, the message is queued, waiting that the receiver agent to read it.

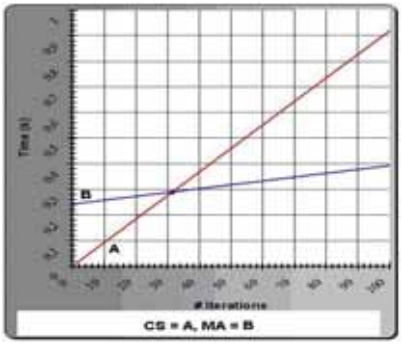
The middleware provides two primitives that support agent mobility [3]: *Relocate* and *Visit*. First of all, when an agent wants to migrate across the network it creates the backpack containing the data to be carried to the destination. Afterwards, when a Relocate is invoked, an agent indicates to the middleware the destination site through its identifier IDsite. If the Visit primitive is invoked, the agent visits the whole network, site after site. It is not necessary to specify any destination but it still necessary to prepare the backpack containing the data. In this case, inside the backpack there is also additional information written by every playground visited by the agent during the Visit. At the implementation level, when a Relocate is invoked, an XML string representing the agent is sent to the destination playground, where a new process running the agent is started, with the backpack as a parameter.

[3] As we already mentioned, we support weak mobility, as defined in [5].

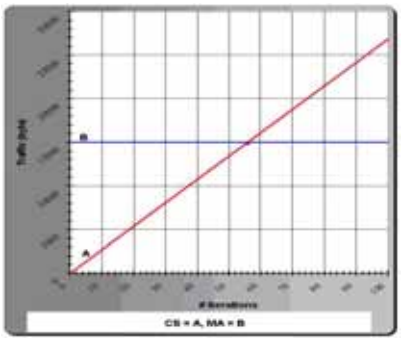
The playground offers to the agents a number of support tools, which allow new NM functionalities to be prototyped. The tools API provides the mechanisms (communication, migration and synchronization primitives) needed to implement different architectural styles; in particular, CS and MAs. Thus through the workbench it is possible to develop, test and monitor applications for NM designed in different paradigms. Different solutions may be tested in the form of mockups at design time by using this workbench, to support architectural reasoning, before implementing them as a new NM applications.

3.1. Experimenting with the Workbench

Hereafter we illustrate how the workbench can be used to support architectural reasoning. In our case study, we analyze the network traffic around the NM control station and we compare CS and MA solutions for a specific network management function that performs *alarm correlation*. Alarm correlation is a useful NM feature, which is often used for alarm filtering or diagnosis. An experiment we made is based on the approach described by [12], which uses neural networks for this task. Because a considerable amount of data must be collected from the network nodes to train the neural network algorithm, we decided to compare alternative solutions based on CS and MA paradigms, by evaluating the traffic generated around the NM control station. The neural network used in our case study needs about 10000 iterations to converge during the training phase. Using our prototype workbench, the solution of our case study can be designed with a CS architecture in which the neural network is implemented by an agent installed on the NM control station. The agent builds the dataset through requests to agents installed on the individual network nodes. Alternatively, it is possible to exploit MAs. Indeed, the neural network can be embodied inside an agent and the learning algorithm is executed on every node. For space reasons, we cannot report the details of the experimental results. However, they show that the CS solution generates much more traffic than the MA solution. Indeed, the latter does not need to transmit the entire dataset, because it exploits its capability to migrate the code near the data. Using MAs, the control station is only in charge of injecting the agent inside the network and receiving it at the end. Figure 4(a) summarizes the training time of the neural network depending on the number of iterations. Figure 4(b) summarizes the network traffic generated around the control station during the training phase. The MA paradigm becomes convenient with a relatively low number of iterations. In particular, it generates a constant traffic (corresponding to the agent implementing the neural network). Vice-versa, the network traffic generated by the CS paradigm grows linearly with the number of iterations.



(a) Time analysis



(b) Traffic analysis

Figure 4. Experimental Analyses

4. Conclusions and Future Work

In this paper we addressed the issue of systematically supporting the choice of an architecture among a set of alternatives through rigorous, quantitative reasoning. We addressed the specific domain of NM, focusing on network traffic minimization. We developed two complementary approaches to architectural reasoning: one based on analytical modeling, and another based on execution of mockups. To support the latter, we developed a workbench environment. The workbench supports fast development of complex NM functionalities by exploiting the abstractions of agents, playgrounds, and tools. Mockup applications can be written in any scripting language and the system is designed with a plug-in support to add new languages and new tools. Finally, we have shown how the workbench can be used to perform experimental analyses. This work is part of our long-term research that aims at supporting software engineers in the design and early validation of software architectures. We are working on a variety of architectural paradigms and a variety of validation approaches covering both functional and non-functional requirements. In particular, we focus on validation techniques that include formal analysis via model checking [3, 10].

References

[1] I. Adhicandra, C. Pattinson, and E. Shaghouei. Using Mobile Agents to Improve Performance of Network Management Operations. *Proceedings of the Post Graduate Network Conference (PGNet 2003)*, 2003.

[2] M. Baldi and G. Picco. Evaluating the tradeoffs of mobile code design paradigms in network management applications. *Software Engineering, 1998. Proceedings of the 1998 International Conference on*, pages 146–155, 1998.

[3] L. Baresi, C. Ghezzi, and L. Mottola. On accurate automatic verification of publish-subscribe architectures. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 199–208, Washington, DC, USA, 2007. IEEE Computer Society.

[4] A. Bieszczad, B. Pagurek, and T. White. Mobile Agents for Network Management. *IEEE Communications Surveys*, 1(1):2–9, 1998.

[5] A. Carzaniga, G. Picco, and G. Vigna. Designing distributed applications with mobile code paradigms. *Software Engineering, 1997., Proceedings of the 1997 (19th) International Conference on*, pages 22–32, 17-23 May 1997.

[6] J. Case, M. Fedor, M. Schoffstall, and J. Davin. RFC1157: Simple Network Management Protocol (SNMP). *Internet RFCs*, 1990.

[7] G. Cugola, C. Ghezzi, G. Picco, and G. Vigna. Analyzing mobile code languages. In *MOS '96: Selected Presentations and Invited Papers Second International Workshop on Mobile Object Systems - Towards the Programmable Internet*, pages 93–110, London, UK, 1997. Springer-Verlag.

[8] T. C. Du, E. Y. Li, and A.-P. Chang. Mobile agents in distributed network management. *Commun. ACM*, 46(7):127–132, 2003.

[9] G. Goldszmidt, Y. Yemini, and S. Yemini. Network management by delegation: the mad approach. In *CASCON '91: Proceedings of the 1991 conference of the Centre for Advanced Studies on Collaborative research*, pages 347–361. IBM Press, 1991.

[10] F. He, L. Baresi, C. Ghezzi, and P. Spoletini. Formal Analysis of Publish-Subscribe Systems by Probabilistic Timed Automata. *LECTURE NOTES IN COMPUTER SCIENCE*, page 247, 2007.

[11] A. Koliousis and J. Sventek. A trustworthy mobile agent infrastructure for network management. *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 383–390, May 21 2007-Yearly 25 2007.

[12] E. Marilly, A. Aghasaryan, S. Betge-Brezetz, O. Martinot, and G. Delegue. Alarm correlation for complex telecommunication networks using neural networks and signal processing. *IP Operations and Management, 2002 IEEE Workshop on*, pages 3–7, 2002.

[13] J. Schönwälder. Network management by delegation from research prototypes towards standards. *Comput. Netw. ISDN Syst.*, 29(15):1843–1852, 1997.

[14] S. Waldbusser. RFC1757: Remote Network Monitoring Management Information Base. *Internet RFCs*, 1995.

PROTEF: Automatic Verification of pattern-based LTL Templates

Luis Garcia*, Steve Roach**, Salamah Salamah***
*IBM Systems and Technology Group.
**Computer Science Department, University of Texas at El Paso.
***Department of Computer and Software Engineering, Embry-Riddle Aeronautical University.

Abstract

Most formal software verification techniques are based on formal specifications of software behavior. Approaches to facilitate the creation of formal specifications include the Specification Pattern System (SPS) and Composite Propositions (CPs). Recent research into generating Linear Temporal Logic (LTL) formulas from SPS patterns resulted in a set of templates that support CPs, but are complex and difficult to verify. This paper describes PROTEF, a software framework to automatically generate and test formulas representing software specifications using model-checker-based testing. This method can be used to test templates in LTL and other formalisms. The framework was used to test LTL templates developed to support CPs.

1. Introduction

Software verification is a fundamental part of the software production process. Formal verification techniques, such as theorem proving [14], runtime-monitoring [4, 5], and model checking [1, 2, 8] are based on formal specifications of software behavior. Creating and validating formal specifications is a significant impediment to the adoption of formal verification techniques [6, 7].

There have been successful research efforts to minimize the challenges of creating formal specifications including the Specification Pattern System (SPS) [3], Composite Propositions (CPs) [10, 11], and the Property Specification Framework (Prospec) [10, 12]. These approaches assist a user in the creation of specifications based on commonly used patterns.

This work introduces the Property Testing Framework (PROTEF), a software framework to automatically generate and test formulas representing software specifications, in particular, specifications based on SPS and CPs. The framework consists of three main components: (1) Property Generator (PROGENE), (2) Property Test Generator (PROTÉGÉ), and (3) Property Tester (PROTEST). The purpose of each component in the framework is to complete each of the processes in the testing method's workflow. *Model checker-based testing* [15] is used here as a general method for testing software specifications based on SPS patterns and CPs. It can be used to test specifications generated in Linear Temporal Logic (LTL) and other formalisms such as Computational Tree Logic (CTL) [9]. PROTEF is described in details in Section 2.4, while PROGENE and (PROTÉGÉ) are described in sections 3 and 4 respectively.

2. Background

2.1. Linear Temporal Logic (LTL)

LTL is a highly-expressive formal language that is widely used in model checkers such as NuSMV [1] and SPIN [8], as well as for the runtime verification of Java programs [18]. The standard LTL operators are shown in Table 1. One problem with LTL is that it is hard to read. For example, it is not immediately obvious that the LTL specification $\square(a \rightarrow \diamond(p \wedge \diamond(\neg p \wedge \neg a)))$ represents the English requirement "If a train is approaching(a), then it will be passing(p), and later it will be done passing with no train approaching".

In this paper, an *execution trace* is represented by positions, read left to right, where each position indicates a state. The spaces are filled with the atomic propositions that are true in that state. A dash indicates no proposition is true. If more than one proposition is true, they are written between parentheses. Table 1 lists the usual LTL operators with example and counter example traces.

2.2. Specification Pattern System (SPS)

Dwyer [3] conducted a survey of software projects and identified a set of commonly occurring software properties. These properties were generalized, formally described, and categorized into *specification patterns*. SPS facilitates the specification of software properties by providing a set of general templates with natural language descriptions. After reading the English descriptions, practitioners identify

Table 1. Description of LTL Operators

Operator	Name	Example Trace	Counter Trace
$\neg$	Not a	$- - - - -$	$a - - - -$
$\wedge$	a And b	$(ab) - - - -$	$- - (ab) - -$
$\vee$	a Or b	$a - - - -$	$- - b - -$
U	a Until b	$a\,a\,a\,b-$	$a\,a\,a - b-$
X	Next a	$-a - --$	$a - - - -$
$\diamond$	Eventually a	$- - a - --$	$- - - - -$
$\Box$	Always a	$a\,a\,a\,a\,a\,a\,a\,a$	$a\,a\,a\,a - a\,a\,a$

Table 2. Description of CP Classes in LTL

CP Class	LTL Description (P^{LTL})
$AtLeastOne_C$	$p_1 \vee \ldots \vee p_n$
$AtLeastOne_E$	$(\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge ((\neg p_1 \wedge \ldots \wedge \neg p_n)\, U\, (p_1 \vee \ldots \vee p_n))$
$Parallel_C$	$p_1 \wedge \ldots \wedge p_n$
$Parallel_E$	$(\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge ((\neg p_1 \wedge \ldots \wedge \neg p_n)\, U\, (p_1 \wedge \ldots \wedge p_n))$
$Consecutive_C$	$(p_1 \wedge X(p_2 \wedge (\ldots (\wedge X p_n)) \ldots))$
$Consecutive_E$	$(\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge ((\neg p_1 \wedge \ldots \wedge \neg p_n)\, U\, (p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n \wedge X(p_2 \wedge \neg p_3 \wedge \ldots \wedge \neg p_n \wedge X(\ldots \wedge X(p_{n-1} \wedge \neg p_n \wedge X p_n)) \ldots))$
$Eventual_C$	$(p_1 \wedge X(\neg p_2\, U\, (p_2 \wedge X(\ldots \wedge X(\neg p_{n-1}\, U\, (p_{n-1} \wedge X(\neg p_n\, U\, p_n)))) \ldots))))$
$Eventual_E$	$(\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge ((\neg p_1 \wedge \ldots \wedge \neg p_n)\, U\, (p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n \wedge ((\neg p_2 \wedge \ldots \wedge \neg p_n)\, U\, (p_2 \wedge \neg p_3 \wedge \ldots \wedge \neg p_n \wedge (\ldots \wedge (p_{n-1} \wedge \neg p_n \wedge (\neg p_n\, U\, p_n)) \ldots)))))$

the template they need and substitute their propositions into the template. SPS introduces the concepts of *pattern* and *scope*. A pattern is the specification template describing the software behavior, and scope is the extent of program execution over which the behavior described by the pattern must hold.

2.3. Composite Propositions (CP)

Some applications require the specification of sequential and concurrent behavior, and it may be necessary to define groups of propositions and the relations between the propositions. Mondragon et al. [10, 11] created a set of abstractions that describe the relation between groups of propositions, referred to as CPs.

Mondragon defined eight CP classes that can be used to describe sequential and concurrent software behavior by identifying the relationships among the propositions in the properties. With CPs, propositions represent either conditions or events. Events are either the beginning or the end of a condition and are instantaneous, while conditions have duration. Conditions describe concurrent behavior, and events describe synchronization. Table 2 provides a description in LTL of the CP classes described by Mondragon. The subscript at the end of the CP class specifies whether the class refers to a set conditions (C) or events (E).

To motivate the use of CPs, consider the example requirement "Request (R) always triggers Acknowledgment (A) between Beginning of execution (B) and System shutdown (S)". Suppose R not only triggers A, but it also triggers Logging (L), and Validation (N). Questions that arise include Do they happen at the same time? Are they all required? Must they occur in a certain order? These concerns are addressed by CPs. In this case, assume that A, L, and N must all occur at the same time. Then we can use the $Parallel_E$ CP class. Using the LTL templates of the CP classes in Table 2 and applying direct substitution of the propositions in the property to the template yields: $Parallel_E$(A, L, N) $\equiv (\neg A \wedge \neg L \wedge \neg N) \wedge ((\neg A \wedge \neg L \wedge \neg N)\, U\, (A \wedge L \wedge N))$.

SPS uses direct substitution of single propositions into predefined templates to generate pattern-based specifications. Salamah [16] demonstrated that it is not possible to use direct substitution of composite propositions in SPS templates without losing the original intent of the pattern and scope.

To address the problem created by direct substitution, Salamah [16, 17] generated a new set of general pattern and scope templates like the templates generated by Dwyer. The correctness of a template is defined as the quality of maintaining its original pattern and scope semantics when substituting CPs while at the same time preserving the original intent of the CP classes. For the purposes of this work, we will call the pattern and scope templates created by Salamah *abstract templates*. Abstract templates (as opposed to SPS templates) have placeholders for CPs rather than single propositions. A template is instantiated by replacing the placeholder in the abstract template with a CP. An abstract template is correct if for all possible instantiations of the abstract template, the meaning of its pattern, scope and CP classes is preserved.

3. Property Testing Framework (PROTEF)

The goal of the work described in this paper is to gain confidence in Salamah's abstract pattern-based templates for LTL by testing them. Testing all the abstract templates is not an easy task given the total number of combinations of patterns, scopes and composite propositions classes. For example, restricting ourselves to CPs with three propositions, the abstract template from the $Absence$ pattern and the $After\ L$ scope, $\neg((\neg L)\, U\, (L \wedge \diamond P))$, would produces a total of 64 formulas, by substituting the 8 existing CP classes for L and the eight existing CP classes for P. Continuing in this fashion for all the abstract LTL templates and all the CP classes would produce over 30,000 formulas. In order to test all of the concrete formulas it is necessary to

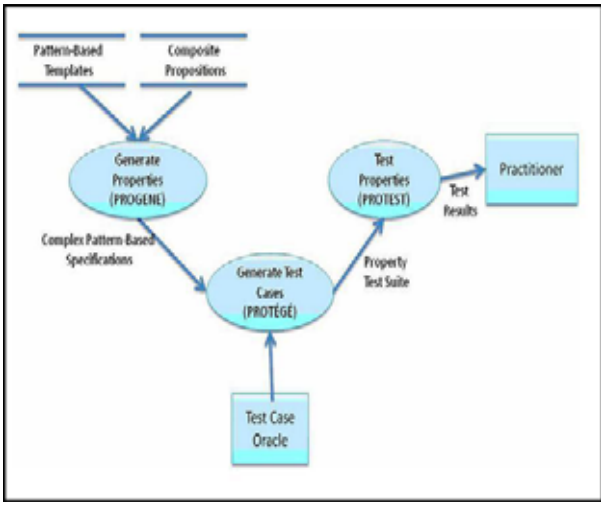


Figure 1. PROTEF Framework

develop a testing method that automates the generation and testing of the templates.

A framework for manipulating properties called Property Manipulation Framework (PROPMAN) was developed that includes the basic object-oriented modules required to manipulate pattern-based properties using Java. The Property Testing Framework (PROTEF) was implemented using PROPMAN. PROTEF is designed to use pattern-based properties for two particular purposes: (a) generating concrete specifications using abstract templates and (b) testing the generated concrete specifications. The effectiveness of PROTEF and the usefulness of our testing approach was demonstrated by testing the pattern-based specifications in LTL for all formulas in the $Absence$ pattern and the $Before\ R$ scope, limiting the choice of composite propositions to $Eventually_E$ and $AtLeastOne_C$.

PROTEF includes three subsystems: the Property Generator (PROGENE), the Property Test Generator (PROTÉGÉ) and the Property Tester (PROTEST). PROGENE is responsible for generating all the possible permutations of specifications that will be tested. PROTÉGÉ is the subsystem responsible for generating test cases for each pattern-based specification. Once we have the specifications and test cases, the job of PROTEST is to perform the actual testing of the formulas. This workflow, devised for testing a large set of pattern-based specifications, is depicted in Figure 1.

3.1. The Property Generator: PROGENE

PROGENE is the framework module that generates the pattern-based specifications that must be tested. Abstract templates are instantiated with CPs producing pattern-based specifications. By using all of the abstract templates for all patterns and scopes, and concrete CPs for all CP classes, we can generate combinations of pattern-based specifications. In the work described here, we have limited the abstract templates to a single pattern, a single scope, and two composite propositions classes.

The Property Generator module has several features required to make the framework general for further use. First, PROGENE works independently of the formal language. Second, it is extensible: In the event that new patterns and scopes are identified, they can be easily included in the list of supported patterns and scopes. Third, it extends to new CPs: Currently the work in composite propositions includes a total of eight composite propositions classes. If new composite propositions classes or new propositions types are identified, they too can be easily incorporated.

We use the abstract LTL templates presented in Salamah [16, 17]. We have chosen to work with only the abstract templates pertaining to the $Absence$ pattern and four CP classes. In the templates, each letter R, Q, L, and R with superscript LTL refers to a placeholder for a composite proposition in LTL, which is to be replaced by the practitioner's own CPs. Salamah's templates use the letters P and R with subscript H to indicate a special class of CPs that refer to the part of the CP where the proposition holds. Salamah also introduces special AND symbols: $\&_r$, $\&_l$ and $\&_{-l}$. These operations do not extend regular LTL, and they have special semantics that make specification easier. The semantics of these operators are out of the scope of this paper; however, PROGENE supports these operators.

PROGENE takes as input the templates similar to the ones shown in Table 3. Each CP placeholder in the templates needs to be identified. A notation for identifying placeholders in the templates was developed:

- $Proposition$: A proposition is a character identifier. The character is one of L, P, Q, or R.
- $NEGATED\_proposition\_type$: This placeholder will be substituted by the negated LTL formula for the CP in $proposition\_type$, where $proposition\_type$ is defined as in $CP(proposition\_type)$.
- $HOLD\_proposition\_type$: This is placeholder represents a special CP class, defined by Salamah. This special CP class is also depicted by an LTL formula and is dependent upon $proposition\_type$, where propositions are defined as in $CP(proposition\_type)$.

PROGENE provides the flexibility for the user to define new placeholders in addition or replacing the ones used to generate the formulas described here and included in the framework by default.

In the rest of this section, the listings, PC, RC, PE, and RE, refer to CPs named P or R, for conditions (C) or events (E) using the definition for $proposition\_type$ given above.

Table 3. LTL Templates for Absence of P Before R

Pattern/Scope	LTL Template
$Absence\,of\,P_C$ $Before\,R_C$	$\neg((\neg R)\,U\,((P\,\&_r\neg R)\,\&_l\,\diamond\,R))$
$Absence\,of\,P_C$ $Before\,R_E$	$(\diamond R \rightarrow \neg((\neg(NEGATED\_R$ $\wedge\,X\,HOLD\_R))\,U\,P\,\&_r\,\neg HOLD\_R)))$
$Absence\,of\,P_E$ $Before\,R_C$	$\neg((\neg R)\,U\,((P)\,\&_r\,\neg R)\,\&_l\,\diamond\,R)$
$Absence\,of\,P_E$ $Before\,R_E$	$(\diamond R \rightarrow \neg((\neg(NEGATED\_R\wedge$ $X\,HOLD\_R))\,U\,P\,\&_r\,\neg HOLD\_R)))$

The first template can be applied to cases where P is of type C. Similarly the second template can be applied to the cases where P is of type E.

The CP templates shown in Table 2 are instantiated using three simple propositions, and the resulting CPs will be directly substituted in the corresponding placeholders in the abstract templates in Table 3. Given the templates and CPs, PROGENE generates all the possible concrete pattern-based properties that can be generated by substituting the composite propositions into the corresponding placeholders in the abstract templates. In addition to the actual specification, PROGENE generates metadata regarding the origin of each concrete pattern-based property. The following listing is an example of the generation of one pattern-based property in our case study:

- Original Abstract Template: "Absence of PC Before RC"
  $\neg((\neg CP(RC))\,U\,((CP(PC)\&_r\neg CP(RC))\&_l\ \diamond\ CP(RC)))$
- Comprising propositions for the P and R CPs:
  - $AtLeastOne_C\ (P_1 \vee P_2 \vee P_3)$
  - $AtLeastOne_C\ (R_1 \vee R_2 \vee R_3)$
- Generated LTL Specification:
  $\neg((\neg(R_1 \vee R_2 \vee R_3))\,U\,(((P_1 \vee P_2 \vee P_3) \wedge \neg(R_1 \vee R_2 \vee R_3)) \wedge \diamond(R_1 \vee R_2 \vee R_3)))$
- Generated Metadata:
  - Base Property:
    - Label: Absence of P_C Before R_C
    - Pattern: Absence of P
    - Scope: Before R
    - Template: $\neg(((\neg R)\,U\,P\,\&_r\,\neg R)\,\&_l\,\diamond\,R)$
  - Generated Property: $\neg((\neg(R_1 \vee R_2 \vee R_3))\,U\,(((P_1 \vee P_2 \vee P_3)\,\&_r\,\neg(R_1 \vee R_2 \vee R_3))\,\&_l\,\diamond\,(R_1 \vee R_2 \vee R_3)))$
  - CP components:

1. Label: $AtLeastOne_C$
   CP Class: At Least One Condition
   Formula: $(P_1 \vee P_2 \vee P_3)$

2. Label: $AtLeastOne_C$
   CP Class: At Least One Condition
   Formula: $(R_1 \vee R_2 \vee R_3)$

The metadata will be used in the testing phase. It includes information such as the CP classes of each of the CPs that comprise the formula, as well as pattern and scope information of the pattern-based specification.

3.2. Property Test Generator (PROTÉGÉ)

A test case for a specification consists of an execution trace and an expected result. PROTÉGÉ is the subsystem that provides the functionality required to manage property tests. This subsystem prepares all the necessary elements for the Property Tester. Our testing method is based on the work of Salamah et al. [15].

Patterns, scopes, and CP classes have specific logical definitions and semantics that can be inferred from their English descriptions. For example, given a pattern-based property of type $AbsenceofPbeforeR$, where P and R are CPs of class $AtLeastOne_C$, we can infer that no CP P is TRUE before the CP R is TRUE. Since both P and R are of type $AtLeastOne_C$, we can further reason that we do not expect even a single proposition that belongs to P to become TRUE, before any of the propositions that belong to R becomes TRUE. This can be visualized using execution traces, for example, "$----R---P-------$". Since P and R are CPs of type $AtLeastOne_C$, if the proposition sets P and R are defined as P = $\{p_1,\ p_2,\ p_3\}$ and R = $\{r_1,\ r_2,\ r_3\}$, then, instantiating the CPs based on the LTL definitions gives $AtLeastOne_C$(P) = $p_1 \vee p_2 \vee p_3$ and $AtLeastOne_C$(R) = $r_1 \vee r_2 \vee r_3$. The original trace of execution may be replaced by "$----r_1-r_2-r_3---p_1-p_2-p_3-----$".

The trace of execution shown above satisfies the property, since there is an absence of all the propositions in P before the first proposition in R appears. Given a trace of execution and a pattern-based property as inputs, the expected output is whether the trace of execution satisfies the property. This constitutes a property test case: a trace of execution, a pattern-based property and TRUE or FALSE, depending on whether the trace of execution should satisfy the property.

Generating Test Cases PROTÉGÉ's main function is to generate property test cases. PROTÉGÉ is designed to automatically generate large sets of test cases. In order to gener-

Table 4. Traces of Computation of Absence P(Eventually$_E$) Before R(Eventually$_E$)

Trace of Computation	Expected Result(T/F)
$(R_1R_2R_3) - -P_1 - -P_2 - P_3 -$	TRUE
$(R_1R_2R_3P_1P_2P_3) - - - - - -$	TRUE
$- - - - - - - - - (R_1R_2R_3)$	TRUE
$- - - - (R_1R_2R_3P_1P_2P_3)$	TRUE
$- - R_1 - R_2 - R_3 - - P_1P_2P_3 -$	TRUE
$- - R_1R_2 - P_1P_2 - R_3 - P_3 - -$	TRUE
$- - R_1R_2R_3 - - P_1P_2P_3R_1R_2R_3 - -$	TRUE
$- - P_1 - P_2 - P_3 - - P_1P_2 - P_3 - -$	TRUE
$- - P_1P_2P_3 - - - R_1R_2R_3 - -$	FALSE
$- - - - - - - P_1P_2P_3R_1R_2R_3$	FALSE
$- - - P_1P_2P_3R_1R_2R_3 - - -$	FALSE

ate tests for all properties resulting from the possible combinations of patterns, scopes, and CPs, it is necessary to use basic execution traces describing pattern-based properties of single propositions. These traces are defined in Salamah et al. [15, 16]. They define the basis for testing all the possible combinations of properties with CPs. In defining these basic traces, Salamah used boundary value analysis and equivalence class testing.

For the purposes of this case study, we generated a set of 68 cases to test the properties of type *Absence of P before R*, for the $Eventually_E$ and $AtLeastOne_C$ CP classes. [1] PROTÉGÉ reads traces and expected results from a file and stores these in the *property test oracle*, which provides tests for a specific pattern-based property. When a request is made to the oracle, it encapsulates the property, the execution trace, and the expected result into a structure that can be used by PROTEST. Table 4 shows example traces of computations used to test the template for $Absence$ of P_E. Before R_E, where both P and R are of type $Eventually_E$.

3.3. Property Tester (PROTEST)

PROTEST executes property tests created by PROGENE and PROTÉGÉ by using a model checker. In order to apply model checking to software three things are needed: a finite state machine model of software, a specification, and a model checker. The model checker exhaustively searches all possible states in the model verifying that it satisfies the specifications. In typical use, a model of a system is built, the system's specifications are created, and a model checker is run to verify that the model satisfies the specifications. Practitioners assume the correctness of the specifications and verify the software model.

[1] The automatic generation of test cases for the remaining pattern/scope combinations is left as a future work, and it is based on the same approach used for generating this set of 68 cases.

```
1.  MODULE main
2.  VAR
3.      Q : seq();
4.  DEFINE
5.      P1 := (Q.State = 3);
6.      P2 := (Q.State = 4);
7.      R1 := (Q.State = 9);
8.      R2 := (Q.State = 10);
10. LTLSPEC !((!(R1 | R2))U((((P1 | P2)&!(R1 | R2))&(F (R1 | R2)))))
11. MODULE seq()
12. VAR
13.     State: 0..14;
14.     ASSIGN
15.         init(State):= 0;
16.         next(State):= case
17.                         (State != 14) :
18.                         {State + 1};
19.                         (State = 14) :{14};
20.                     esac;
```

Figure 2.
SMV Code Used in Testing

In our case, rather than verifying the model, we assume the correctness of the model and verify the specifications. This is the basis for what we call *model-checker-based testing* of properties. PROTEST includes an interface to the NuSMV model checker [1]. NuSMV was chosen for the simplicity of its modeling language, its graphical user interface, and its availability. By providing an interface to NSMV, PROTEST users are able to run the NSMV model checker using Java.

An execution trace can be transformed into a model in the following manner. Given an example execution trace: $- - - P_1P_2P_3 - - - R_1R_2R - - -$, number the states starting at 0. The example trace shows that P1 is true only in state 3, p2 is true only in state 4, and so on. We generate a model that has only one possible execution path. Figure 2 shows the SMV code created for this example.

The SMV code creates a Boolean variable for each of the propositions in the trace of execution and determines the value of the counter variable when the propositions will be true. The 'LTLSPEC' defines a property specification. The 'ASSIGN' section creates a counter from 0 to 14 matching exactly the trace of execution. A model is generated for each test case, the model checker is invoked, and the result is compared to the expected result.

4. Example Template Verification

To demonstrate the PROTEF framework, we generated and tested a subset of all the possible pattern-based properties based on SPS and CPs. The subset of properties included the *Absence of P* pattern and the *Before R* scope, limiting ourselves to two choices of composite propositions classes $AtLeastOne_C$ and $Eventually_E$.

The subset of the properties that we used included four pattern-based CPs. Table 5 shows the results of executing the model checker-based testing for the four generated prop-

Table 5. Test Results For Absence Before R

P	R	Test Cases	Tests Passed
$AtLeastOne_C$	$AtLeastOne_C$	16	16
$AtLeastOne_C$	$Eventually_E$	20	20
$Eventually_E$	$AtLeastOne_E$	18	18
$Eventually_E$	$Eventually_E$	14	14

erties: $Absence$ of P_C Before R_C, $Absence$ of P_E Before R_C, $Absence$ of P_C Before R_E, and $Absence$ of P_E Before R_E.

The initial execution of the test suite generated for the *Absence of P before R* pattern produced several failures. Expert reviews with the author of the templates, temporal logic experts, and practitioners revealed three primary causes: (1) There were errors in the expected values supplied to PROTÉGÉ. (2) The original templates were incorrect. (3) There were misinterpretations of the semantics of the $\&_r$ operation introduced in Salamah [16]. After the problems were corrected (i.e., the templates were updated and the $\&_r$ operator reimplemented), the test cases were re-executed successfully.

These three classes of errors demonstrate the importance of the testing effort. We independently discovered errors in the original templates, the same errors discovered during Salamah's attempts at formal correctness proofs. The miscalculation of the expected value of the test cases indicates how difficult it can be to create and read complex specifications. Tool support is essential. It is important to consider these original test failure causes in order to avoid repeating these potential failures in the future.

5. Conclusions and Future Work

One of the goals of this project is to be able to verify the correctness of all the pattern-based properties for any pattern, scope and CP class. In this work we built the software framework needed to achieve that goal and we demonstrated how to use the framework to verify a subset of the properties. The SPS authors state that they verify the correctness of their properties through formal peer reviews. PROTEF could help in the verification of their formulas in a more systematic and autonomous way and provide a higher degree of reliability.

The generation of execution traces is a fundamental part of testing the properties that is not automatic. It may be possible to create general tests from which concrete tests based on specific composite propositions classes can be generated, and further work is required in this area.

References

[1] Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M., "NuSMV: a new Symbolic Model Verifer" International Conference on Computer Aided Verifcation CAV, July 1999.

[2] Clarke, E., Grumberg, O., and D. Peled. Model Checking. MIT Publishers, 1999.

[3] Dwyer, M. B., Avrunin, G. S., and Corbett, J. C., "Patterns in Property Specification for Finite-State Verification," Proceedings of the 21st Intl. Conference on Software Engineering, Los Angeles, CA, USA, 1999, 411-420.

[4] Gates, A., and Roach, S., "DynaMICs: Comprehensive Support for Run-Time Monitoring," in Proceedings of the Runtime Verification Workshop 2001, Paris, France, Vol. 55, No. 2, July 2001, pp. 61-77.

[5] Gates, A., and Mondragon, O., "A Constraint-Based Tracing Approach," Journal of Systems and Software, 2002.

[6] Hall, A., "Seven Myths of Formal Methods," IEEE Software, September 1990, pp. 11-19.

[7] Holloway, M., and Butler, R., "Impediments to Industrial Use of Formal Methods," IEEE Computer, April 1996, pp. 25-26.

[8] Holzmann, G. J., "The model checker SPIN" IEEE Transactions on Software Engineering., 23(5):279-295, May 1997.

[9] Laroussinie, F. and Ph. Schnoebelen, "Specification in CTL+Past for verification in CTL," Information and Computation, 2000, 236-263.

[10] Mondragon, O., "Elucidation and Specification of Software Properties through Patterns and Composite Propositions to Support Formal Verification Techniques" Dissertation, Computer Engineering Department, University of Texas at El Paso, 2004.

[11] Mondragon, O. and Gates, A., "Supporting Elicitation and Specification of Software Properties through Patterns and Composite Propositions," Intl. Journal Software Engineering and Knowledge Engineering, XS 14(1), Feb. 2004.

[12] Mondragon, O., Gates, A., and Roach, S., "Prospec: Support for Elicitation and Formal Specification of Software Properties," in Proceedings of Runtime Verification Workshop, ENTCS, 89(2), 2004.

[13] Prior, A. N., Past, Present and Future. Oxford: Clarendon Press, 1967.

[14] Rushby, J., "Theorem Proving for Verification," in F. Cassez, (Eds.): M odeling and Verification of Parallel Processes, Springer-Verlag, Nantes, France, 2000.

[15] Salamah, S., Gates, A., Roach , S., and Mondragon, O., "Verifying Pattern-Generated LTL Formulas: A Case Study. Proceedings of the 12th SPIN Workshop on Model Checking Software. San Francisco, California, August, 2005, 200-220

[16] Salamah, S., "Generating Linear Temporal Logic Formulas For pattern-based Specifications" Dissertation, Computer Science Department, University of Texas at El Paso, 2007.

[17] Salamah, S., Gates, A., Kreinovich, V., and Roach, S., "Using Patterns and Composite Propositions to Automate the Generation of Complex LTL", Proceedings of the 5th International Symposium on Automated Technology for Verification and Analysis ATVA, Tokyo, Japan, October 22-25, 2007 pp. 533(9).

[18] Stolz, V., and Bodden, E., "Temporal Assertions using AspectJ", *Fifth Workshop on Runtime Verification* Jul. 2005.",

Formal Specification of Object-Oriented Systems with Collaborative Objects and Petri Nets – A Case Study

Boleslaw Mikolajczak[1,2]
[1]*Computer and Information Science Department*
University of Massachusetts Dartmouth, MA 02747, USA
[2]*Polish-Japanese School of Information Technology, Warsaw, Poland*
bmikolajczak@umassd.edu

Abstract

This paper presents a case study of integration of object-oriented design with Petri nets. Cooperating Objects are used for Petri net modeling of the object-oriented design. Firstly, all objects are represented by separate Petri nets. Secondly, a coordinating Petri net is designed to represent collaboration of objects involved. Benefits of both methodologies and of Object-Oriented Design and formalisms of Petri nets are used for verification and validation purposes. ATM case study has been used to illustrate the method presented. We used SYROCO tool in process of rapid prototyping of Object-Oriented systems by means of Petri nets.

1. Introduction

The purpose of this paper is to explore the integration of Object Oriented Design (OOD) using Petri nets (PN). OOD techniques are well established, but they lack a formalism and rigor. This becomes apparent when attempts are made to verify and validate a design prior to implementation. Extending the OO methodology to include parallelism makes the problem worse.

PNs provide a well understood formal method for modeling concurrent systems. PNs do not support the concepts of modularization, encapsulation, and information hiding. It is highly desirable to find a way to combine the best characteristics of these two differing design methods. In [4], a strategic amalgamation of the OOD with PNs has been presented. The approach preserves results of OOD and allows verification, and validation of the designed system using PN-based tools.

There are several approaches to integration of OOD and PNs. Among them are: Sibertin-Blanc [5], and Ceska et al. [3]. The goal of this paper is to explore the integration of OOD technique, PN's formalisms, and methodology of Cooperative Objects [5]. The case study chosen was that of a bank ATM machine. This OOD is adopted from [8] and it is not presented in the paper. We selected SYROCO that provides technology platform to define Cooperative Objects (COO). The COO language uses PNs to model internal operation of objects, and provides a mechanism to connect together the PNs of distinct objects.

The rest of the paper is organized as follows. In section 2 we present details of transformation from OOD to Collaborative Objects and to respective Petri nets. This section presents also execution of several scenarios for the ATM machine. The paper presents only small portion of a much larger project in technology transfer.

2. Petri Net Representation of the ATM Object-Oriented Design

We define first general algorithmic strategy of OOD specification with transformations into Petri nets that is expressed through the following macro-steps:

Step 1. For each class/object of the OOD define a corresponding Petri nets with its Object-Control Structure (OBCS). Classes/objects with inherent concurrency should have this feature explicitly expressed in the PN model.

Step 2. Using Collaboration Diagram of the OOD, create one PN that represents formalization of coordination of all classes/objects of the OOD by means of Petri net.

Step 3. Verify/Validate the OOD through simulation of the PN model; this will involve execution of the PN model with various positive and negative scenarios of interest.

We implemented *UserInterface* subsystem of the ATM, and a skeleton of the *Atm* class. The following objects were implemented as Cooperative objects: *Atm, BankCardReader, Form, Menu, SecureForm*, and *UserMessage*. Taking into account space limitation of the paper we present two PN models only. Fig. 1 represents a PN model of initialization of the *Atm* with a selection of ATM operations. Fig. 2.presents a PN model of the card validation process.

2.1. The Petri net model of the Atm class

The *Atm* class is the root class of the system. The *Atm* COO object is initialized when *Atm Init* operation executed. The *Atm Init()* operation creates the *BankCardReader* device object and initializes it. It also initializes the *Greeting* object. ATM system main menu is then initialized. The main menu is an instance of the *Menu* class. Finally, a single token is deposited into place *Initial* in the *Atm* OBCS.

A token in place *Initial* (Fig. 1) enables the firing of transition *InitializeAtm*. This transition invokes service *AtmInit*. Completion of service *AtmInit* deposits a token in place *Start*. A token in place *Start* enables transition *WaitForCustomer*. This transition invokes the *InsertValidCard()* service of the Greeting object, which is of class *UserMessage*. Upon completion of the service call, a Customer Identifier (cid) is returned, which becomes the token deposited in the *HaveCustomer* place. This token enables the *DisplayMenu* transition, which invokes the *GetChoice()* service of the *mainMenu* object, an instance of the Menu class. Upon completion of the *GetChoice()* service, a menu selection is returned which is deposited into place *HaveSelection* as a token. The menu selection token in place *HaveSelection* enables one of the five transitions connected to the place. Each transition handles one of the functions available on the main menu of the ATM system.

A valid menu selection of type QUERY enables the *BalanceInquiry* transition. This invokes a stub operation in the *Atm* class that reports activation of the Balance Inquiry transaction. Completion of this operation moves the menu selection token back into the *HaveCustomer* place, enabling the *DisplayMenu* transition as before. A valid menu selection token of type DEPOSIT in place *HaveSelection* enables the *Deposit* transition. This in turn invokes a stub operation in the *Atm* class that reports activation of the *Deposit* transaction. Completion of this operation moves the menu selection token back into the *HaveCustomer* place, enabling the *DisplayMenu* transition as before. A valid menu selection token of type TRANSFER in place *HaveSelection* enables the *Transfer* transition. This invokes a stub operation in the *Atm* class that reports activation of the Funds Transfer transaction. Completion of this operation moves the menu selection token back into the *HaveCustomer* place, enabling the *DisplayMenu* transition.

A valid menu selection token of type DONE, or an invalid menu selection representing activation of the *CancelKey*, in place *HaveSelection* enables the *Completed* transition. This in turn invokes a stub operation in the *Atm* class that reports completion of the customer session. Completion of this operation moves the menu selection token into the *Finish* place. The token in the *Finish* place enables the *RdyNextCustomer* transition. This transition invokes the *RemoveCard()* service of the Greeting object (from the *UserMessage* class). Upon completion of the *RemoveCard()* service, the return value is passed as a token to the *ResetATM* place. The token in the *ResetATM* place enables the *Restart* transition. This transition invokes the *LogUser()* operation, which is a stub simulating the activities performed to close out a customer session. Upon completion of the *LogUser()* operation, a token is deposited in the *Start* place, readying the *Atm* object for another customer session. The *AtmInit()* service makes a series of calls to the *AddItem()* service of the *MainMenu* object. These calls initialize the ATM main menu. Each call provides a text string used as the menu label, and a *menuChoice* item returned when that menu entry is selected.

2.2. The UserMessage Design

An arbitrary number of *UserMessage* objects can be instantiated at any time, each one represented by independent PN Object Control Structure (OBCS). The *Greeting* object is defined by the *Atm*. The object is initialized by *Init()* operation that appears in the *UserMessage*. Each *UserMessage* object requires access to single *BankCardReader* object and single *DisplayScreen* object of the ATM system Fig. 2).

The *Greeting* object *InsertValidCard()* service is invoked by transition *WaitForCustomer* of the *Atm* object. This invocation enables transition t1 of the *UserMessage* OBCS. Transition t1 deposits the service request token in place *SrvReq*, and it deposits a control flow token in place *GreetingMsg*. A token in place *GreetingMsg* enables transition t2. When t2 fires, it calls the *DisplayScreen* device, and then it invokes the *Input()* service of the *BankCardReader* device. Transition t2 does not complete until the *BankCardReader Input()* service completes, returning a *UsrResp* token (Rtn) which is deposited in place *CardStatus*. This token indicates whether or not a valid bankcard has been inserted into the ATM machine and validated by customer. A valid response token in the *CardStatus* place, along with a service request token in the *SrvReq* place, enables transition t3. This transition returns the response (Rtn) to the caller that invoked the *InsertValidCard* service. In the case of a valid response, the return value contains customer identification (cid). An invalid response token in the *CardStatus* place indicates an invalid card, or a cancel request by the customer. In this case transition t4 is enabled, which deposits a token into the *InsertWaiting* place. The *InsertWaiting* place will not enable transition t5 until the token has been in place for a time delay. After that, transition t5 can fire and will deposit the token into place *GreetingMsg*. This restarts the greeting cycle again, the original request remains pending and control does not pass back to the caller of the *InsertValidCard()* service.

The Greeting object service *RemoveCard()* is invoked by the ATM object OBCS transition *RdyNextCustomer*. This invocation enables transition t6 of the *UserMessage*

OBCS. Transition t6 deposits the service request token (*req*) into place *Srv2Req* and also deposits a control token into place *RemoveRdy*. The token in the *RemoveRdy* place enables transition t7. Transition t7 invokes the *Eject()* service of the *BankCardReader* device. Upon returning from the *Eject()* service, a token is deposited into place *Ejected* of the *Greeting* object. The presence of a token in place *Ejected*, as well as the presence of a service request token (*req*) in place *Srv2Req* enables the firing of transition t8. Transition t8 creates a return parameter (Rtn) and sets it to a valid response before returning it as a token to the caller of the *RemoveCard* service.

2.2. The *BankCardReader* Design

Startup of the *BankCardReader* Object OBCS (Fig. 2). Only one *BankCardReader* object is created in the ATM system. This occurs during initialization of the *Atm* object. A pointer to this object is saved and distributed to other classes that require access to the *BankCardReader*. The *BankCardReader* device is initialized by the *Atm* object invoking its *Init()* routine. The *BankCardReader Init()* operation first initializes the *Form* object called *pinForm*. This is a secure form used for entry of the customer PIN number. Then the prompt used to obtain the PIN is initialized. The *BankCardReader* object OBCS begins running as a result of the call to the *Init()* operation.

The *Input()* service of the *BankCardReader* is invoked by transition t2 of the *Greeting* object service *InsertValidCard*. This call enables transition t1 of the *BankCardReader* OBCS. The action of transition t1 calls operation *IsCardPresent()*. This operation is a stub. Transition t1 uses the boolean flag to fire one of two emission rules depending on the returned value. If there is no card present in the device, transition t1 deposits a token in place *NoCard*. If there is a card present in the device, then a token is deposited in place *CardPresent*. In either case, a token *req*, representing the service request, is also deposited in place *SrvcReqHolding*.

The token in place *CardPresent* enables transition t2. Transition t2 calls operation *IsCardValid()*. This operation is a stub that simulates a hardware/software function that determines whether or not the inserted card is a valid ATM bankcard. If the inserted card is not valid, transition t2 deposits a token in place *InvalidCard*. This token enables transition t3, which displays an appropriate message to the *DisplayScreen* device, and puts a token into place *RemoveCard*. Operations following place *RemoveCard* will be described later. If the inserted card is valid, transition t2 transfers a token to place *ValidCard*, which subsequently enables transition t6. Transition t6 calls operation *ReadCardPin(),* followed by a call to operation *ReadCardCid()*. Operation *ReadCardPin()* is a stub. The stub operates by asking the user to enter the PIN number associated with the card. The response is saved in attribute *cpin* of the *BankCardReader* object.

2.3 The *Menu* Design

An arbitrary number of *Menu* objects may be created at any time. Each object includes two basic functions, an *AddItem* service that builds the menu by adding selectable menu items to it, and a *GetChoice* service that displays the menu in its current state, allowing selection of an entry and returning a *UsrResp* item indicating the selection. The design of the Menu COO class contains by far the most complicated OBCS of any of the other COO classes. The implementation included one instance of a Menu object, the *mainMenu* object used by the *Atm* subsystem as its main menu for a customer to choose a transaction to be performed. The complexity of this class demands that an overview of the design be presented prior to describing the detailed functioning of the Petri Net composing the OBCS. The *AddItem* service is used to build the menu, adding entries to the *MenuData* place. This place is the central data store that holds the essence of the menu.

The design approach implemented was to map the current set of menu choices (it starts from the beginning of the menu) from place *MenuData* onto the available set of keypad keys stored in place *KeyData*, creating a set of menu display items for the current page of the menu, which are then stored in place *DisplayData*. This set of display items contains additional menu choices not present in place *MenuData* to specify selection of the next menu page, or a return to the top of the menu if we are at the last menu page. The current page of the menu is then displayed one item at a time, constructing a set of menu choice items, which are stored in place *ChoiceData*. A keypad entry is solicited from the customer and used to select a choice token from place *ChoiceData*. This item is either returned as a menu selection, or used to generate the next display page of the menu if it is a "next page" or a "top of menu" selection. The handling of boundary conditions adds to the complexity of the Menu object. Cases in which the menu has no entries (the *AddItem* service has not yet been called), there are more keys available than menu items to be displayed, or a key is pressed that does not correspond to a valid menu choice must be properly handled. The Menu class design selected allows for a menu to change (via the *AddItem* service) between invocations of the *GetChoice* service.

The *mainMenu* object is declared by the *Atm* object. The *mainMenu* object is initialized with a call to its *Init()* service. The Menu class *Init()* service begins by saving the pointers to the *DisplayScreen* and the *Keypad* objects passed to it. Then a number of attributes are initialized to denote an empty menu. The text labels used to display the "next page" and the "top of menu" entries are defined. The keys available for menu selection use are defined in

array *key_data*. The *KeyItem* class is defined as X. Tokens representing these key definitions are created and put into place *KeyData*. The initialized contents of place *KeyData* must be preserved throughout all of the menu processing. At this point the *mainMenu* object OBCS begins running, and control returns to the *Atm* object. The *AddItem()* service is used to add menu entry items to the menu object. Examples of *AddItem()* service calls for the *mainMenu* object can be found in transitions *InitT1* through *InitT5* of the *AtmInit* service. Parameters of the service call are the label, the text string used to label the menu entry, and the choice, a *menuChoice* type, used to denote the menu item selected. An invocation of the *AddItem()* service enables transition t1 of the Menu OBCS. The transition calls operation *SaveItem()*. This operation creates a *MenuItem* from the label and choice input parameters. A *MenuItem* consolidates the menu entry text label, the return choice, the index, or position of this menu entry within the menu, and the key that will be assigned to this entry. Transition t1 deposits this *MenuItem* as a token into place *MenuData* and returns control to the caller of service *AddItem()*. Place *MenuData* holds the essential data comprising the menu. The contents of this place must persist uncorrupted throughout the lifetime of the menu object.

2.4. Startup of the ATM

The *Atm* object is designated as the *root* object of the system. This part shows the ATM waiting for insertion of a card into the bank card reader device. The "no" answer given to the *BankCardReader* stub operation *IsCardPresent* simulates the hardware device sensing the absence of a card. After a short delay imposed by the *Greeting* object the main menu greeting prompt reappears on the display screen.

This part demonstrates the customer canceling during PIN entry. The ATM machine detects the presence of a valid, readable bankcard. The bankcard reader device then extracts the PIN number and the Customer Identification number from the bankcard. Next the customer is prompted to enter the PIN. The customer enters the correct PIN, but before pressing the ENTER key, decides to cancel by striking the CANCEL key. Note that the "x" key as described in the design of the *Keypad* class simulates the CANCEL key. As a result, the card is ejected and the ATM machine resets for the next customer. A valid card is inserted and the PIN and Customer ID are read from it. The customer is prompted to enter the PIN the first time, and it is entered incorrectly. Notice the secure version of the *Form* class echoes the PIN entry back so it is unreadable on the display screen. An appropriate response is displayed to the customer and he is given another opportunity to enter the PIN. This is repeated two more times until the ATM machine announces it is retaining the bankcard. At this point the bank card reader device would "swallow" the ATM card, the machine resets, and the initial greeting message is again presented. Successful validation of the customer makes available the customer ID for use by the rest of the system. The customer elects to terminate the session by selecting the completion item on the menu. The system announces completion of the transaction, termination of the customer session, ejection of the card and resets with display of the main greeting message. Notice that removal of the card in this scenario is different than in previous scenarios. In previous scenarios the card is never validated and so the *InsertValidCard* service of the *Greeting* object never returns control to the *Atm* object. In this scenario the *Greeting* object does complete and the *Atm* object executes the *mainMenu* object. The card removal is done by invocation of the *Greeting* object *RemoveCard* service [transaction *RdyNextCustomer* in the Atm OBCS.

3. Conclusion

The basic concept to combine OOD with Petri Nets to model causality and concurrency is sound. The concept is to embed the Petri net inside the object class, using it to model the current state of the object. It also ties together all of the objects Petri nets into a structure that can be run for simulation purposes. The paper demonstrates that effective and scalable application of this concept to a middle size OOD is practically possible.

4. References

[1]. Agha G., de Cindio F., Rozenberg G. (Eds.), *Concurrent Object-Oriented Programming and Petri Nets*, LNCS 2001, Springer, 2001.

[2]. Buchs D., and Guelfi N., *A Formal Specification Framework for Object-Oriented Distributed Systems*, IEEE Trans. on SE, Vol 26, No. 7, July 2000.

[3]. Ceska M., Janousek V., Vojnar T., *PNtalk - A Computerized Tool for Object Oriented Petri Nets Modelling,*, LNCS 1333, Springer, 1997, 591-610, http://www.fee.vutbr.cz/~janousek/pntalk/node23.html

[4]. Mikolajczak, B., Mukhin, D., *A Method of Concurrent Object-Oriented Design Using High-Level Petri Nets,* Proc. of the Int. SMC'98 Conference, San Diego, Oct. 1998.

[5]. Sibertin-Blanc C., Hameurlain N., Touzeau P., *SYROCO: A C++ Implementation of Cooperative Objects,* Proc. of the Workshop Petri Nets and Object-Oriented Models of Concurrency, Turin, June 1995.

[6]. http://www.univ-tlse1.fr/ceriss/COOgene.html

[7]. http://www.daimi.au.dk/~petrinet/tools/

[8]. Wirfs-Brock R., Wilkerson B., and Wiener L., *Designing Object-Oriented Software*, Prentice-Hall, 1990.

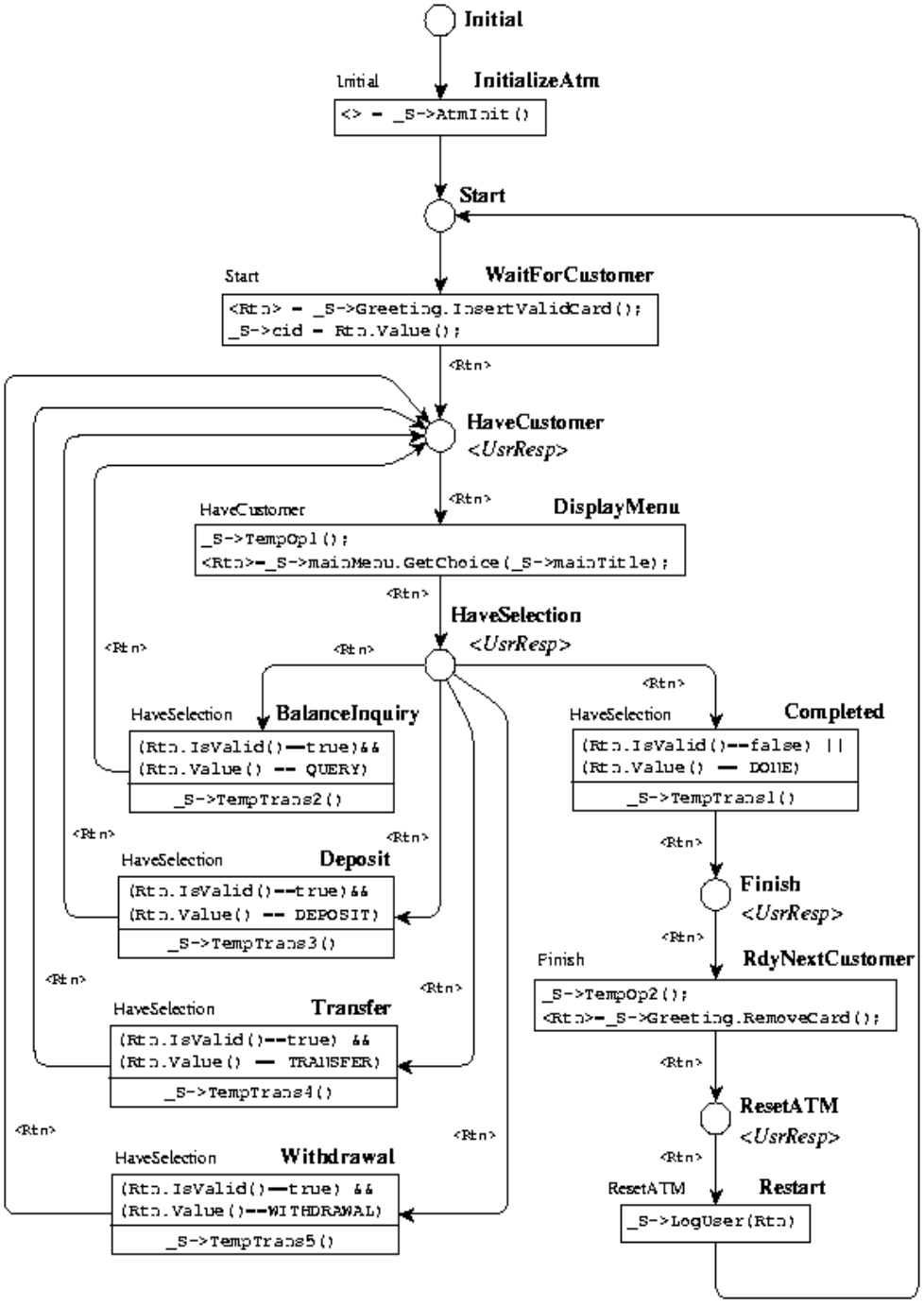


Fig. 1. Petri net model of the root *Atm* OBCS.

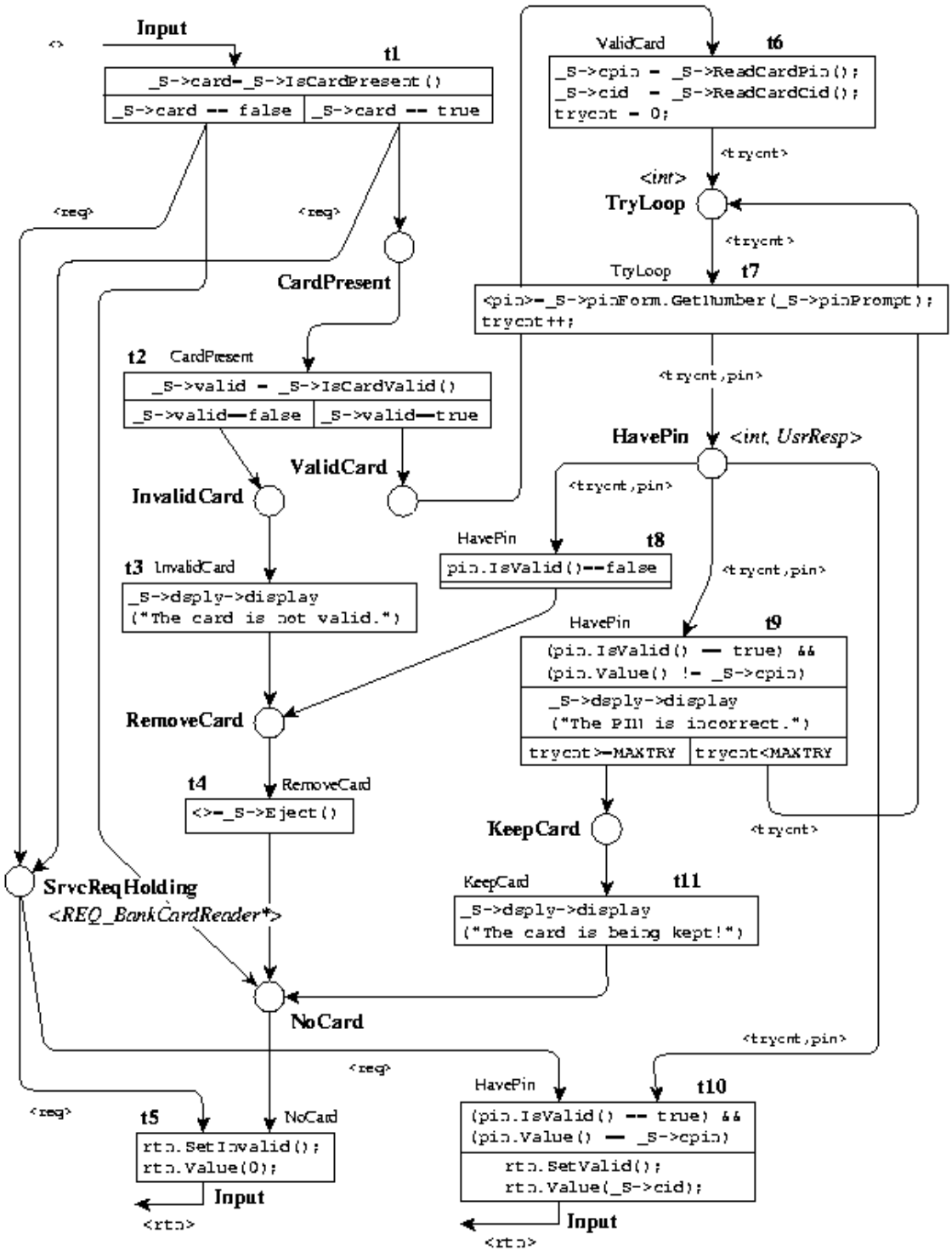


Fig. 2. Petri net model of the Input service.

A Property Specification Tool for Generating Formal Specifications: Prospec 2.0

Irbis Gallegos[1], Omar Ochoa[1], Ann Gates[2], Steve Roach[2], Salamah Salamah[3], Corina Vela[1]

[1,2] Department of Computer Science, The University of Texas at El Paso, El Paso, TX 79968, USA

[3] Department of Computer and Software Engineering, Embry Riddle Aeronautical University, Daytona Beach, FL 32114, USA

[1]{irbisg, omar, cyvela}@miners.utep.edu, [2]{agates, sroach}@utep.edu, [3]salamahs@erau.edu

Abstract

Numerous formal approaches to software assurance are available, including: runtime monitoring, model checking, and theorem proving. All of these approaches require formal specifications of behavioral properties to verify a software system. Creation of formal specifications is difficult, and previously, there has been inadequate tool support for this task. The Property Specification tool, Prospec, was developed to assist users in the creation of formal specifications. This paper describes Prospec 2.0, an improvement to the previous version, by addressing the results of a study conducted to assess the usability of the tool and by adding functionality that supports the validation process.

1. Introduction

Formal methods to support software assurance require the identification of behavioral properties of the software system, generation of formal specifications for the properties, validation of the specifications, and verification of the correctness of the system. The effectiveness of the assurance approach depends on the quality of the formal specifications, and a significant hurdle to the use of formal approaches is the development of correct formal specifications.

Typically, the person creating the formal specification must have a strong mathematical background and be aware of the subtleties of the specification language. For example, model checkers, such as SPIN [1] and NuSMV [2] use formal specifications written in Linear Temporal Logic (LTL) [3], which can be difficult to read, write, and validate. This problem is compounded if requirements must be specified in more than one formal language, which frequently is the case if more than one verification tool is used. The specifier must be aware of the differences in expressiveness of each of the target languages.

The Property Specification (Prospec) 1.0 tool was developed to address some of these challenges. Prospec uses the Specification Pattern System (SPS) [4] and Composite Propositions (CP) [5] to assist developers in the elicitation and specification of system properties.

Usability studies of Prospec have shown that it facilitates the elicitation, understanding, and specification of formal properties [6].

This paper describes Prospec 2.0. In particular, it describes the new features in Prospec that are aimed at improving the tool's support for generating and validating formal property specifications.

2. Background

The Specification Pattern System (SPS) [4] is a set of patterns used to assist in the formal specification of properties for finite-state verification tools. SPS patterns are high-level abstractions providing descriptions of common properties that hold on a sequence of conditions or events in a finite state model. SPS patterns characterize two behavioral aspects: the occurrence and the order of events or conditions.

Occurrence patterns are *universality*, *absence*, *existence*, and *bounded existence*. Order patterns are *precedence*, *response*, *chain of precedence* and *chain of response*. In SPS, a chain pattern defines a sequence of events or conditions. *Chain-precedence* and *chain-response* patterns permit specifying a sequence of events or conditions as a parameter of precedence or response patterns, respectively. SPS restricts the specification of sequences to precedence and response patterns.

In SPS, a pattern is bounded by the scope of computation over which the pattern applies. The beginning and end of the scope are specified by the conditions or events that define the left (L) and right (R) boundaries, respectively.

A study by Dwyer et. al. [4] identified the *response* pattern as the most commonly used pattern, followed by the *universality* and *absence* patterns. These three patterns accounted for 80% of the 580 properties sampled in the study. Because of the frequency with which response properties occur, it is important to provide abstractions that support multiple propositions when specifying sequence of events or concurrent behavior. Because multiple propositions may occur in the cause and effect part of response properties, CPs can be used to assist in their specification and validation. By using CPs in either part of the response pattern (the cause or effect), it is possible to represent common behavior associated with

concurrent systems, such as synchronized join and fork, concurrency, non-determinism, and sequences.

Mondragon et al. [5] introduced Composite Propositions (CPs) to handle pattern and scope parameters that represent multiple conditions or events. The introduction of CPs supports the specification of concurrency, sequences, and non-consecutive sequential behavior on patterns and scope. Mondragon proposes a taxonomy with twelve classes of CPs. In this taxonomy, each class defines a detailed structure for either concurrent or sequential behavior based on the types of relations that exist among a set of propositions.

The original version of Prospec is an automated tool that guides a user in the development of formal specifications. It includes patterns and scopes, and it uses decision trees to assist users in the selection of appropriate patterns and scopes for a given property. Prospec 1.0 extends the capability of SPS by supporting the specification of CP classes for each parameter of a pattern or scope that is comprised of multiple conditions or events. The use of CP classes allows practitioners using Prospec to specify ordered sequences, non-deterministic sequences, and concurrency. By using CPs, a practitioner is directed to clarify requirements, which leads to reduced ambiguity and incompleteness of property specifications.

Prospec uses guided questions to distinguish the types of scope or relations among multiple conditions or events. By answering a series of questions, the practitioner is lead to consider different aspects of the property. A type of scope or CP class is identified at the end of guidance. Prospec generates formal specifications in Future Interval Logic (FIL) [7] and the Meta-Event Definition Language (MEDL) [8].

3. PROSPEC 2.0

3.1 Prospec Revisions

A formal experiment evaluated the effects that the original Prospec and SPS have over the quality of the generated software property specifications with respect to completeness and correctness [17]. SPS supports the creation of specifications through a web site and manual substitution of propositions into templates. The following research hypothesis was supported: *users who specify software properties using Prospec, identify, on the average, more correct patterns and scopes than users who specify software properties using the SPS web site.* The subjects also provided comments for Prospec in the post-evaluation form. They suggested that Prospec:

- provide the capability to access all the properties defined in a given project;
- allow the capability to apply the negation operator to propositions;
- indicate the properties that contain a recorded assumption; and
- modify the physical position and labels for parameters *S* and *P* in the *response* and *precedence* patterns in the pattern screen.

These and other observations made when using Prospec motivated the creation of Prospec 2.0. The revised tool includes changes to the user interface and, more significantly, the tool is being revised to generate LTL specifications with support for validation of the specifications. To provide the ability to export properties into other software tools, Prospec 2.0 uses XML.

3.2 Linear Temporal Logic Generation

Salamah et. al. [9, 16] showed that direct substitution of one or more parameters for a pattern or scope that includes CPs may result in a specification that does not meet the intent of the user. Consider the following example: "The delete button is enabled in the main window only if the user is logged in as administrator and the main window is invoked by selecting it from the Admin menu." The property could be classified *Existence(P)* with *Before R* scope, and *P* is a classified as $Eventual_c(p_1,p_2)$[*1] where p_1 denotes "User logged in as an administrator," p_2 denotes "Main window is invoked," and *R* denotes "Delete Button is enabled. The SPS template for *Existence(P) Before R* is $(\Diamond R) \rightarrow (!R \; U \; (P \wedge !R))$ and the formula for $Eventual_C$ is $(p_1 \wedge X\,(!p_2 \; U \; p_2))$. Direct substitution would yield:

$$\Diamond R \rightarrow (!R \; U \; (!R \wedge (p_1 \wedge X\,(!p_2 \; U \; p_2)))).$$

This, however, would permit the delete button to be enabled between the time that the administrator logs in and the administrator invokes the main window.

To address this, Salamah [16] introduced general templates to support the generation of LTL formulas that use CPs. For example, consider the *Response (P, Q)* pattern with Global scope in which *P* and *Q* are $Consecutive_C(p_1, p_2)^*$, i.e., $(p_1 \wedge X\,(p_2))$ and $Parallel_C\,(q_1, q_2)^*$, i.e., $(q_1 \wedge q_2)$. The general template for the "Response- Global scope" is:

$$\Box\,(P^{LTL} \rightarrow (P^{LTL} \;\&_l\; \Diamond \;\; Q^{LTL})),$$

where P^{LTL} and Q^{LTL} represent LTL formulas for the CP class and $\&_l$ is a special operator that ensures that $\Diamond\, Q^{LTL}$ is "anded" only with the last element of the sequence represented by P^{LTL} [9, 16]. As a result the formula becomes:

$$\Box\,((p_1 \wedge X\,(p_2)) \rightarrow (p_1 \wedge X\,(p_2 \;\wedge\; (\Diamond\,(q_1 \wedge q_2))))).$$

[*] The complete list of CP classes and their LTL descriptions is available in Mondragon et. al., [5].

In the previous example for enabling the delete button, the general template for the *Existence(P)* with *Before R* scope example described earlier is: $!((!(P^{LTL} \&_r\ !R^{LTL}))\ U\ R^{LTL})$, where $\&_r$ is a special operator that denotes that $!R^{LTL}$ holds at all the states within the sequence represented by P^{LTL} [9, 16]. As a result, the correct formula is:

$$!((!(((p_1 \ \&\ !R)\ \&\ X((!p_2\ \&\ !R)U\ (p_2\ \&\ !R))))\ U\ R)$$

To support validation of generated formulas, each general formula has associated traces of computations that represent behaviors that are accepted or not accepted by the LTL formula being tested. The traces of computation can be used to test the formulas using a model checker [16, 20]. In addition to supporting the process of testing the LTL formulas, these templates provide visual representations to aid the user in understanding the meaning of the complex LTL formulas that are generated, helping those users who are not immersed in formal representations.

Salamah [15] used multiple techniques to verify the correctness of the general formulas: formal proofs, testing, and reviews. In order to support the formal proofs, the work included formal definitions of patterns and scopes that use CPs. This supports the ability to define similar properties in other specification languages.

A secondary effort in Salamah's work involved modifying the original LTL formulas provided by Prospec for patterns and scopes whose parameters contained only single propositions and CP. The approach to simplify the formulas was to reduce the number of states in the Büchi automaton (BA) generated from an LTL formula. The size (number of states) of the automaton that results from the intersection of the BA generated by the LTL formula and that of the system model has as its upper bound the product of the number of states in each of the two. Work has been done by other researchers on the translation of LTL to BA to reduce the number of states in the resulting BA and to speed up the process of the BA generation [12, 13, 14]. It was possible to reduce the number of temporal operators, and as a result improve the efficiency of 17 out of 30 the formulas defined by Prospec [21].

3.3 Interface

Prospec 2.0 includes new features to support the specification of properties. The interface maintains the support and functionality of the original Prospec, i.e., the guiding screens for selecting scope and patterns remain the same. The main changes to the interface are related primarily to CP specifications and information presentation.

Figure 1. presents the main screen for Prospec 2.0. To the left in the main screen the *Property Browsing Tree* is shown. The larger frame on the right encloses screens (as opposed to the original implementation in which individual screens were distributed over the available screen space). The larger frame provides practitioners with a separate context for each property. The enclosing frame enables concurrent property specification as well as easy transition between these property specifications.

Since Prospec 2.0 supports concurrent property specifications, a *Property Browsing Tree* is available for accessing properties. The *Property Browsing Tree* allows practitioners to browse, traverse and quickly preview properties being specified. Also, the *Property Browsing Tree* allows editing of properties attributes such as scope, pattern, CPs, and propositions, as shown in Figure 1. Once a user selects a property attribute in the tree, the appropriate window will be opened allowing modification of the property attribute.

Another interface improvement is the *Visual Representation window*. This window will provide a visual representation of the specified property as a trace of computation that shows the scope and the specified property pattern. The visual representation in conjunction with the written description will be the base for the validation capabilities of Prospec 2.0.

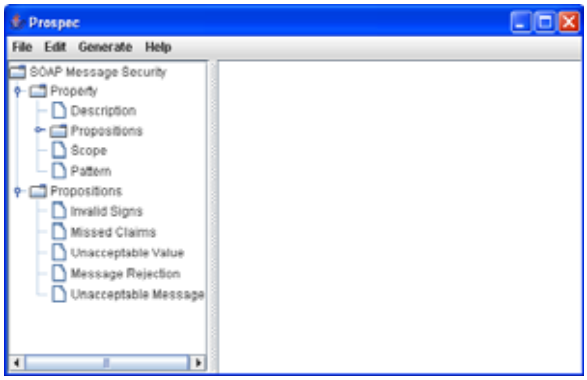


Figure 1. Prospec 2.0 Main Window

The *Property* window describes the basic property information such as the property name, the informal property description as provided by the client and any assumptions made about the property, as shown in Figure. 2 and Figure 3. Properties can be created by clicking on the *New* button, removed by clicking on the *Delete button* and stored into the viewing table by clicking on the *Save* button. Different Properties can be browsed and viewed by selecting them from the table of propositions. The *scope* section includes the *scope* type, assumptions made about the *scope*, and the left and right propositions. CP attributes include a *Type* to differentiate between events and conditions, a *CompositeProp* to identify the desired CP, and a *PropositionList* including the simple propositions to be used in the CPs. A proposition is described by a symbol and a description.

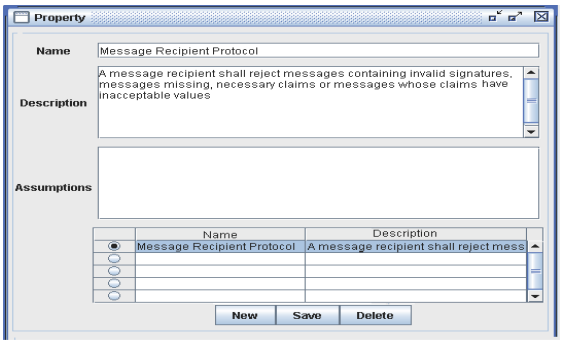


Figure 2. Prospec 2.0 Property Specification Window Upper Section

A new feature in Prospec 2.0 is a window that allows the user to view a summary of the property being specified. The window shows the current state of the specification and the formal specification if defined. This window is embedded into the *Property* window, as shown in Figure 3.

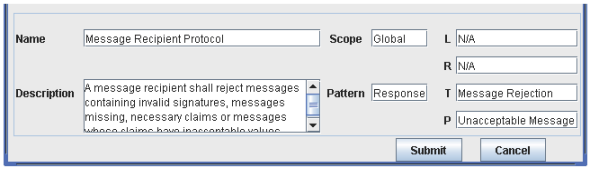


Figure 3. Prospec 2.0 Property Specification Window Lower Section

Prospec 2.0 allows users to create, save, and print reports of the specifications. The reports are created either directly from the Prospec 2.0 application or from the XML files containing the metadata of the property specifications. The reports include the informal specification as provided in the description section of Prospec 2.0, the formal specification as generated by Prospec 2.0, information to construct the visual representation matching the generated formal specification, and the corresponding metadata such as logic used or version number.

4. Scenario

The following scenario illustrates the use of Prospec. Jill is a software engineer working on security issues in web services. She recognizes that the system must support the following requirement: "A message recipient shall reject messages containing invalid signatures, messages missing necessary claims, or messages whose claims have unacceptable values [18]." Since the project team will use a model checker to verify the algorithms, she needs an LTL specification.

Jill starts Prospec and creates the new property project *SOAP Message Security* and selects *LTL* as the logic to be used for the formal specification. She accesses the property browsing tree and double-clicks the *property description* attribute to open the *property description* screen. Using this screen, she names the property *Message Recipient Protocol* and provides the informal description.

Now Jill must identify the scope, the region of the program over which the property must hold. She accesses the property browsing tree and double-clicks the scope attribute to open the scope specification screen, which displays English descriptions of the five available scopes. After using the decision tree in the Guided Selection screen, Jill selects Global as the property scope as shown in Figure 4. Properties with Global scope must hold over all states of execution.

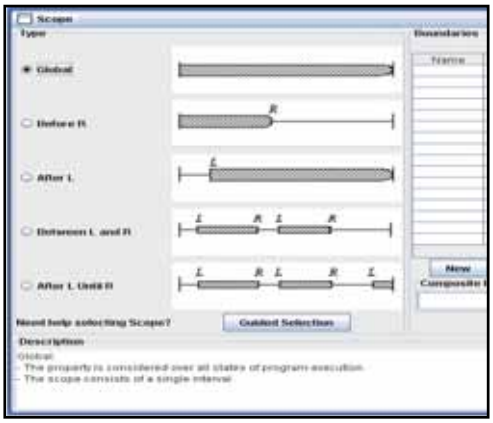


Figure 4. Section of Prospec 2.0 Scope Window

After identifying the scope, Jill selects a pattern. Selecting the *pattern* attribute of the property browsing tree opens the *pattern specification* screen. The five available patterns are described there, and Jill decides to use the *Response (T,P)* pattern as shown in Figure 5. In this pattern, the conditions or events described by *T* are a response to the conditions or events described by *P*. The letters *T* and *P* represent a proposition and a composite proposition (CPs), respectively. This choice is appropriate since Jill wants to ensure that there is a rejection whenever an unacceptable message is received. Prospec offers guided selection to assist a user in the selection of an appropriate pattern.

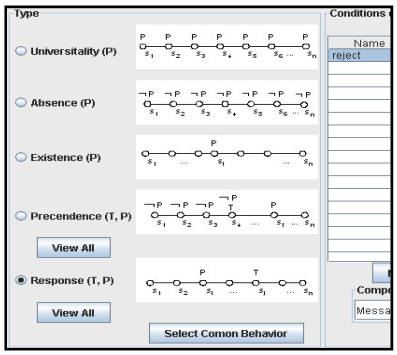


Figure 5. Section of Prospec 2.0 Pattern Window

Once the pattern has been selected, Jill defines *T* and *P*. To define *P*, Jill creates the simple propositions *invalid_sign* (*Message contains invalid signatures*), *miss_claims* (*Messages are missing necessary claims*), and *unacceptable_value* (*Messages have claims with unacceptable values*). Because *P* is defined by three propositions, Prospec displays a message indicating that a CP must be defined. Once the CP screen is accessed as shown in Figure 6, Jill determines that the three simple propositions are *conditions* (*propositions that hold in one or more consecutive states*) and should be combined using $AtLeastOne_C$ *(G)* to indicate that at least one of the propositions in the set *G* can trigger the condition. A decision tree in the *Guided Selection* screen for CPs can be used to determine which CP to use. To define the *T* parameter, Jill creates a new proposition *reject* (*Message is rejected*), indicating that the message recipient rejects the incoming message. This completes the pattern definition.

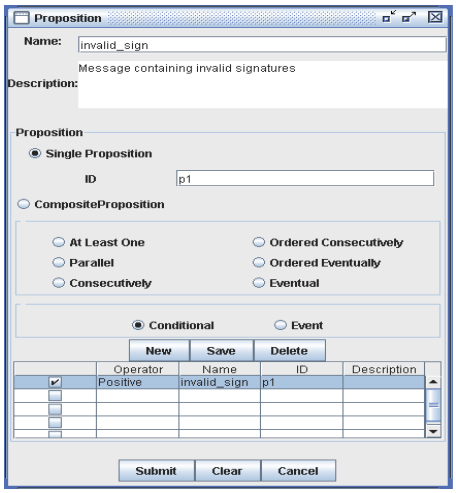


Figure 6. Prospec 2.0 Scope Window

To create the LTL formula from the scope and pattern, Jill selects the *view formula* button on Prospec's main screen. Prospec takes the scope, pattern, and CP and generates the LTL formula:

$$\square((invalid\_sign \mid miss\_claim \mid unacceptable\_value) \rightarrow \Diamond reject)$$

This formula specifies that during the execution of the program, whenever an unacceptable message is received by a message recipient, a rejection follows. In addition to the LTL formula, a set of sample execution traces is presented showing possible sequences of execution and the value of the formula for each sequence. Jill reviews these execution traces to ensure that the formula captures her intent.

An execution trace is a sequence of states read from left to right. Each space represents the propositions that are true in that state. If no proposition is true, a "-" is used. If more than one proposition is true, the propositions are enclosed in parentheses. Some of the execution traces for the *SOAP Message Security* property are shown in Table 1. For this example, symbol *I* stands for proposition *invalid_sign*, symbol *M* for proposition *miss_claim*, symbol *U* for proposition *unacceptable_value*, and symbol *R* for proposition *reject*.

Trace of Computation	Result
--I---R---	*Satisfied*
--RU------	*Unsatisfied*
U---M-----	*Unsatisfied*
(IM)--------R	*Satisfied*

Table 1. SOAP Message Security Property Execution Traces Examples

5. RELATED WORK

This section describes other tools used also for the elicitation and formal specification processes and how these efforts differ from Prospec 2.0.

5.1 Propel

The goal of Propel [10] is to help practitioners write and understand properties by providing templates that explicitly capture details as options for commonly-occurring property patterns based on SPS. The provided templates are represented using both disciplined natural language (DNL) and finite-state automata (FSA). The practitioner can view both representations simultaneously and select from which representation to elucidate the desired property.

The main difference between Prospec 2.0 and Propel is that Prospec 2.0 uses guided questions to distinguish the types of scope or relations among multiple conditions or events while Propel uses DNL. Also the pattern visual representations differ between the efforts, in Prospec 2.0 timelines are used while in Propel FSA are used.

5.2 Timeline Editor

Timeline Editor [11] allows the formalization of certain type of requirements. To formalize these requirements a series of events and required system responses are placed on a timeline. The tool converts the timeline specification automatically into a test automaton. The timeline specification can then be used directly by a logic model checker or a test-sequence generator.

As opposed to Prospec 2.0, Timeline Editor cannot capture group of events occurring in arbitrary order nor provide visual feedback for validation purposes. Prospec 2.0 allows practitioners to specify group of events

occurring in arbitrary order by using CPs and SPS. Also, the traces of computation generated by Prospec 2.0 allow practitioners to validate that the specified properties match the practitioner's intent.

5.3 SPIDER

SPIDER [19] generates specification properties using natural language representations. This process is based on a natural language grammar and specification pattern system to derive a natural language sentence. This sentence is then mapped to the temporal logic that can be analyzed formally by a tool such as SPIN. The structured language grammar supports translations of untimed and timed properties to multiple temporal logics.

There are three main differences between Prospec 2.0 and SPIDER. Prospec 2.0 offers support for composite propositions, guided selection in the specification process, and property validation using traces of computations.

6. CONCLUSIONS

In this paper, we describe Prospec 2.0, an improvement to the property specification tool Prospec 1.0. We discuss the new features of Prospec 2.0 and describe how these changes enable practitioners to use Prospec 2.0 as both as an automated formal property specification tool and as an automated formal property specification validation tool. The use of XML in Prospec 2.0 and its ability to be interoperable, it will be possible now to integrate the Prospec into the chain of tools that could provide the desired end-to-end automation for all aspects of software development.

7. ACKNOWLEDGMENTS

The authors would like to thank Milos Janek and Roberto Nevarez for their work towards this project. This work is partially supported through the CREST Cyber-ShARE Center funded by NSF grant number HRD-0734825.

8. REFERENCES

[1] G. J. Holzmann, "The model checker SPIN." IEEE Trans. on Softw. Eng., 23(5):279--295, May 1997.

[2] A. Cimatti, E. Clarke, F. Giunchiglia and M. Roveri, "NuSMV: a new symbolic model verifier". In *Proceeding of International Conference on Computer-Aided Verification (CAV'99)*. Trento, Italy, July 1999.

[3] Emerson, E., "A temporal and modal logic." In *Handbook of theoretical Computer Science (Vol. B): Formal Models and Semantics*, J. van Leeuwen, Ed. MIT Press, Cambridge, MA. 1990. 995-1072.

[4] Dwyer, M. B., Avrunin, G. S., and J. C. Corbett, "Property specification patterns for finite-state verification." In *Proceedings of the Second Workshop on Formal Methods in Software Practice* (Clearwater Beach, Florida, United States, March 04 - 05, 1998). FMSP '98. ACM Press, New York, NY, 7-15.

[5] Mondragon, O., Gates, A., and S. Roach, "Composite propositions: toward support for formal specification of system properties," *Proceedings of the 27th Annual IEEE/NASA Goddard Software Engineering Workshop*. Greenbelt, MD, USA, December 2002.

[6] Gates, A. Q., Mondragon O., and F. Kassem, "Automated Support for Property Specification Based on Patterns." In *Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering*, July 2003, pp. 174-181.

[7] Kutty, G., Moser, L. E., Melliar-Smith, P. M., Dillon, L. K., and Y. S. Ramakrishna, "First-Order Future Interval Logic." In *Proceedings of the First international Conference on Temporal Logic* (July 11 - 14, 1994).

[8] Kim M., Kannan S., Lee I., and O. Sokolsky, "Java-MaC: A run-time assurance tool for Java*."* In *Proceedings 1st International Workshop on Run-time Verification.* 2001.

[9] Salamah S., Gates A. Q., Kreinovich V., and S. Roach, "Verification of automatically generated pattern-based LTL specifications." In the *Proceedings of the 10th IEEE International Symposium on High Assurance Systems Engineering,Dallas, TX, November,* 2007.

[10] Smith, R.L., Avrunin, G.S., Clarke, L.A., and L.J. Osterweil, "PROPEL: an approach supporting property elucidation." In *Proceedings of the 24rd International Conference on Software Engineering.* 2002, pp. 11-21.

[11] Smith, M. H., Holzmann, G. J., and K. Etessami, "Events and constraints: a graphical editor for capturing logic requirements of programs." In *Proceedings of the 5th IEEE international Symposium on Requirements Engineering* (August 27 - 31, 2001). IEEE Computer Society, Washington, DC, 14-23.

[12] Oddoux, D., and P. Gastin, "Fast LTL to Büchi Automata translation," 13th International Conference on Computer Aided Verification, CAV, Jul. 2001.

[13] Etessami, K., and G. Holzmann, "Optimizing Büchi Automata," *Proceedings of 11th International Conference on Concurrency Theory*, 2000.

[14] Fritz, C., "Constructing Büchi Automata from Linear Temporal Logic using simulation relations for alternating Buchi Automata," Eighth Conference on Implementation and Application of Automata, 2003.

[15] Salamah, I. S., "Defining LTL formulas for complex pattern-based software properties," University of Texas at El Paso, Department of Computer Science, PhD Dissertation, May 2007.

[16] Salamah, S., Gates, A., Roach, S., and O. Mondragon, "Verifying pattern-generated LTL formulas: a case study." *12th International SPIN Workshop*, Aug. 2005.

[17] Modragon, O., "Elucidation and Specification of Software Properties through Patterns and Composite Propositions to Support Formal Verification Techniques," Ph. D. Dissertation, Computer Science Department, University of Texas at El Paso, May 2004.

[18] Organization for the Advancement of Structured Information Standards. Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). *OASIS Standard Specification*. February 2006.

[19] Konrad, S., and B. H.C. Cheng, "Facilitating the Construction of Specification Pattern-based Properties," In *Proceedings of the IEEE International Requirements Engineering Conference (RE05),* August 2005.

[20] Salamah, S. and Gates, A., "A Technique for Using Model Checkers to Teach Formal Specifications" to appear *Proceedings of the Conference of Software Engineering Education and Training,* Charleston, South Carolina, April 2008.

[21] Salamah S., Gates, A., and S. Roach "Improving Pattern Based LTL Formulas for Model Checking" to appear in the *Proceedings of the 5th IEEE International Conference on Information Technology (ITNG)*, Las Vegas, NV, April 2008.

On the Rarity of Fault-prone Modules in Knowledge-based Software Quality Modeling

Taghi M. Khoshgoftaar
Computer Science and Engineering
Florida Atlantic University
777 Glades Rd., Boca Raton, FL 33431
Email: taghi@cse.fau.edu

Naeem Seliya
Computer and Information Science
University of Michigan – Dearborn
4901 Evergreen Rd., Dearborn, MI 48128
Email: nseliya@umich.edu

Dennis J. Drown
Computer Science and Engineering
Florida Atlantic University
777 Glades Rd., Boca Raton, FL 33431
Email: ddrown@fau.edu

***Abstract*—A large imbalance between proportions of the not-fault-prone and fault-prone program modules in a software measurement dataset has an adverse impact on the trained software quality model. This rarity of known fault-prone modules during software quality modeling is a common occurrence, especially in high assurance systems. Such disparity in proportions of the two classes is observed in other domains as well. We present an evolutionary computing-based data sampling approach to address class imbalance in binary classification problems. The approach, named Evolutionary Sampling, works by "naturally" selecting the good instances in the training data and removing those that are redundant, irrelevant, or impart no additional knowledge. We compare our majority undersampling technique with the other existing majority undersampling techniques, i.e. random undersampling, Wilson's editing, and one-sided selection. Our approach can also be combined with a genetic algorithm-based optimization of a classifier's modeling parameters. The multilayer perceptron neural network is used in this study as the underlying software quality model. Case studies of two real-world software measurement datasets are used for evaluating our genetic algorithm-based data sampling approach with the other majority undersampling techniques. It is shown that Evolutionary Sampling is significant in outperforming the other data sampling techniques, and shows a clear improvement over the software quality model built without any data sampling.**

I. Introduction

Knowledge-based software quality modeling often involves learning from a training dataset that consists of a very large number of high-quality (or not-fault-prone) program modules and a very small number of low-quality (or fault-prone) program modules. Often seen in high-assurance systems [1], [2], [3], this rarity of known fault-prone modules poses a unique class imbalance that impedes the training of a useful software quality estimation model. The class imbalance problem is also observed in other domains, such as diagnosis of rare medical conditions [4], detecting oil spills from satellite images [5] and detection of computer security breaches [6].

A software quality classification model is typically built using known software measurement and defect data from a prior system release or a similar software project developed previously. The given classification algorithm aims to extract and learn associations between the different software metrics and the defect data for the training dataset. The validated model is then ready to predict the unknown quality-based class for a program module with known software metrics.

Given a binary classification problem such as predicting program modules as either fault-prone (*fp*) or not-fault-prone (*nfp*), a training dataset that suffers from class imbalance can be divided into a majority class (not-fault-prone) and a minority class (fault-prone). A classifier trained on such a skewed dataset is more likely to predict a new instance as belonging to the majority class since it was over-represented during the training process. Such a model is clearly not useful, since the more important minority class (fault-prone modules) will go undetected more often. The definition of what is considered as a *fp* or *nfp* program module is dependent on the software project and its quality improvement objectives.

We present a novel and effective genetic algorithm-based approach for sampling a training dataset suffering from class imbalance such that instances from the majority class are intelligently removed. The resulting training dataset is relatively more balanced than the original since the minority class size remains unchanged. A data sampling technique that reduces the majority class size, while maintaining the minority class is termed majority undersampling. In contrast, a data sampling technique that increases the minority class size, while maintaining the majority class is termed minority oversampling. Developed as a research prototype, eVANN implements the proposed Evolutionary Sampling (EVS) approach and also facilitates a GA-based optimization of the underlying learner's modeling parameters. eVANN is extensible to almost any existing binary classifier.

This paper compares EVS with every other existing majority undersampling techniques that address class imbalance in machine learning problems. Those techniques include random undersampling [7], Wilson's editing [8], and one-sided selection [9]. The other category of data sampling techniques (i.e., minority oversampling) that address class imbalance are not considered for comparison in this paper due to space limitations; however, they are compared with EVS elsewhere [10].

The EVS data sampling approach is presented and evaluated with case studies of two real-world software measurement datasets. In the case of both case study datasets, the minority class instances (i.e., *nfp* modules) are less than 10% of the total number of instances in the given training dataset. The same software project data are also used in the comparison of EVS with the three other majority undersampling techniques. The Multilayer Perceptron, as implemented in WEKA [11], is the learner used to build software quality models in this study.

We empirically demonstrate that EVS improves classifier performance compared to software quality modeling without applying any data sampling technique. In addition, it is shown that EVS performs significantly better than all other majority undersampling techniques, i.e., EVS is better than random undersampling, Wilson's editing, and one-sided selection for the case studies presented.

The contributions of this paper include: (1) presenting Evolutionary Sampling, a novel GA-based data sampling technique for addressing class imbalance, (2) a computationally intensive empirical study that compares the proposed data sampling approach with other existing majority undersampling techniques, and (3) application of data sampling techniques for addressing rarity of fault-prone modules in knowledge-based software quality estimation modeling.

The remainder of the paper is structured as follows: Section II

summarizes the other three majority undersampling techniques that are compared with EVS; Section III discusses the default parameter settings for the Multilayer Perceptron learner and performance metrics used for evaluating the software quality classification models; Section IV details the proposed data sampling approach along with relevant GA background; Section V describes the case studies, and discusses the empirical results; Section VI concludes our paper with key research findings and suggestions for future work.

II. Other Majority Undersampling Techniques

The three other majority undersampling techniques we examine include random undersampling, Wilson's editing, and one-sided selection. Random undersampling (*RUS*) is an effective, yet simple, technique in which a portion of the majority class instances are removed at random from the training data. Weiss et al. [7] report an equal balance between the majority and minority classes is often desirable, but that the optimal ratio between the two groups will vary with different datasets and domains.

Wilson's editing (*WLE*) strives to remove noisy instances of the majority class based on a k-nearest-neighbor (k-NN, with k = 3) algorithm that classifies each instance in the training dataset using the remaining instances [8], [12]. The training dataset is reduced by removing instances that were incorrectly classified by the 3-NN algorithm. An alternate version of WLE incorporates a weight factor, based on the class distribution, that is taken into account when computing the distance between instances. This weighting results in some bias toward identification of the minority class instances.

One-sided selection (*OSS*) aims to remove both noisy and redundant instances of the majority class from the training dataset [9]. Initially the redundant instances are removed by creating a consistent subset of the original training dataset that will correctly classify all of the training data using a one-nearest-neighbor (1-NN) rule. Incorrectly classified instances and borderline instances which lie close to the boundary between the two classes in the feature space are removed using Tomek links [13]. A Tomek link exists between a positive (minority class) and a negative (majority class) instance if the two instances are nearest neighbors of each another.

III. Selected Learner and Performance Metrics

The learner we use in conjunction with the different data sampling techniques in this study is the Multilayer Perceptron (MLP). We use WEKA's [11] implementation of the MLP learner with (unless stated otherwise) the following default parameter settings: a learning rate of 0.3 with no decay; a momentum rate of 0.2; 500 training epochs/iterations, validation set at 10% of original dataset; an error threshold of 20 for the validation set; a random number generator seed of 0; and 3 nodes in one hidden layer.

In a two-group (positive and negative) classification problem, if the positive group represents the minority class and the negative group represents the majority class, then a false positive indicates an error in which a majority class instance is incorrectly classified as belonging to the minority class [7], [14]. A false negative indicates an error in which a minority class instance is incorrectly classified as belonging to the majority class. A true positive and true negative would respectively represent correct classification of a minority instance and a majority instance. Evaluating competing models with multiple metrics simultaneously may be difficult when there is no clear winner. We use singular metrics that are well-known performance metrics in data mining and machine learning – Area-Under the ROC Curve, Geometric Mean, and F-measure [11].

An ROC (Receiver Operating Characteristic) curve is a visual representation of a classifier's performance in which the model's true positive rate (y-axis) is plotted against its false positive rate (x-axis) [15]. A desirable ROC curve is one that maximizes the Area-Under the ROC Curve (AUC). Since an ROC curve does not show bias toward the majority class, the AUC value serves as a good singular performance metric when dealing with class imbalance. The AUC provides a rating for the classifier's general predictive ability regardless of class prior probabilities and misclassification costs [16].

The Geometric Mean (GMean) provides a singular measure for evaluating classifier performance compared to using multiple classification error rates or classification accuracies. The GMean is computed as $\sqrt{(true\ positive\ rate) \cdot (true\ negative\ rate)}$. Kubat et al. [9] suggest the geometric mean as a good performance metric for classification problems involving class imbalance. A higher GMean indicates that a classifier is balanced and shows good performance for both classes. We use the AUC and GMean performance measures in the fitness function for our Evolutionary Sampling process, as explained in Section IV-B. The AUC and GMean are also computed as model evaluation metrics.

The *F-Measure* (F-Meas) is based on two information retrieval metrics, Recall (or effectiveness) and Precision (or efficiency), where Recall is the true positive rate and Precision is the ratio of the true positive rate and the sum of the true positive rate and the false positive rate. When Recall and Precision are given equal importance, the F-measure is computed as $\frac{2 \times Recall \times Precision}{Recall + Precision}$. The fitness function of our Evolutionary Sampling approach does not include the F-measure, which is instead used as an independent performance metric to evaluate the classifiers.

IV. GA-Based Data Sampling

A. Genetic Algorithms

GAs perform optimization, simulating natural evolution by creating offspring as a result of mating of the fitter individuals [17]. Each individual is a solution to the problem being addressed. This evolutionary process continues until some stopping criterion (e.g., achieved fitness level or number of generations processed) is reached and the fittest individual is chosen and will hopefully have a near-optimal solution encoded in its genes [18]. The genes are the parameters of a solution, i.e. an individual. During the simulated evolution, various genetic operators (e.g., *crossover*, *mutation*, etc.) may be used to mimic the mating process among fitter individuals.

The crossover (or recombination) genetic operator simulates the process of natural mating by creating offspring that contain genetic material from both parents. To create the chromosomes in the children, we set a randomly determined *locus*, or crossover point in the parents' chromosomes. The first child receives a copy of the first parent's genes up to the crossover point and the second parent's genes after that point. The other child receives the second parent's genes to the left of the locus, and the first parent's genes to the right. The above crossover is a single-point crossover, and is used in our study. The other crossover types used in the literature, multi-point crossover and uniform crossover [19], are not used in our study.

The mutation genetic operator plays a key role in the exploration of the search space as it actually alters the gene alleles to new values instead of simply recombining the gene alleles which already exist in the population. When an individual's chromosome consists of a binary string, the mutation operator may invert the bit value for a randomly selected gene. In a more complex chromosome that encodes integer or real values, creep mutation can be used to add or subtract a random amount from the gene, but within specified bounds. We

use a method proposed by Tate [20] which creates mutated clones of individuals independently of the crossover genetic operator.

A natural selection process is needed to determine which individuals in the population have better fitness. The best chromosome selector orders the individuals in the population by fitness and selects the fittest to remain for the next generation. The best chromosome selector can lead to an early convergence which may represent a local-minimum and not the optimal solution [21]. We use the best chromosome selector as a preprocessing stage for the crossover and mutation operators with the aim of preventing a poor individual from being selected for crossover and mutation.

Ranking selection orders the individuals by fitness at the beginning of the selection process, and only considers an individual's goodness ranking relative to other individuals, regardless of their actual fitness magnitudes. Generally, tournament selection is preferred over roulette wheel selection [18]. In tournament selection, a set number of individuals (tournament size) are randomly selected to participate in the tournament. We select four individuals for tournament selection, as explained in the next section. Once the individuals are picked, the system ranks them according to their fitness and then plays out as a tiny roulette wheel. The fitter the individual in the tournament, the greater its chance of being selected. The winner of the tournament survives to become part of the next generation. The system conducts as many tournaments as needed to obtain the desired population size. We use tournament selection at the end of each generation to provide a way to return the population to its desired size for the next generation.

B. Evolutionary Sampling

elÀNN is a GA-based data mining application developed for our research on addressing class imbalance. Written in Java, this tool currently includes three machine learners, Multilayer Perceptrons, C4.5 and RIPPER, and is based on the WEKA framework [11]. While elÀNN can be extended to include Evolutionary Sampling (EVS) with other machine learners, in this study we focus on Multilayer Perceptrons (MLP). elÀNN also implements a GA-based solution for optimizing modeling parameters of the machine learner, i.e., the MLP architecture in our case. EVS is a majority undersampling technique that works to reduce the number of majority class instances used to train a classifier. This philosophy reflects the relatively very large proportion of the majority class (*nfp*) instances as compared to the minority class (*fp*) instances.

We utilize the basic gene types provided in the Java Genetic Algorithms Package (JGAP) library [22] which elÀNN uses as its GA engine. In JGAP, each of the genes (`BooleanGene`, `IntegerGene`, and `DoubleGene`) is an individual object and the set of genes for an individual is contained in a `Chromosome` object. The genes are considered basic units in the chromosome for purposes of crossover and mutation, and we can set maximum and minimum values for each gene. The chromosome of an individual in elÀNN contains information on how to sample the dataset and how to optimize the learner when building the classifier. We use GA to find an optimal sample of the majority classes used in the fold's training portion of the dataset – five-fold cross-validation is used in our study.

We define two basic chromosome parts (see Table I) for an individual in a GA-based experiment using elÀNN. Part A contains the genes needed to tune a machine learner's modeling parameters, and Part B contains the genes used to perform evolutionary data sampling. Both parts of the chromosome can be ignored during evolutionary computing based on whether each is required for a given experiment in our study.

TABLE I
CHROMOSOME DEFINITION FOR THE MLP LEARNER

Part	Gene Type	Values	Description
A	`DoubleGene`	0.0 to 1.0	Learning rate
	`DoubleGene`	0.0 to 1.0	Momentum rate
	`IntegerGene`	1 to $\hat{m}$	Number of nodes in first hidden layer
	`IntegerGene`	0 to 2	Number of additional hidden layers
	`IntegerGene`	1 to $\hat{m}$	Number of nodes in second hidden layer
	`IntegerGene`	1 to $\hat{m}$	Number of nodes in third hidden layer
	`BooleanGene`	true or false	Decay learning rate for back-propagation
B	`BooleanGene`	true or false	Sampling flag for N_0
	`BooleanGene`	true or false	⋮
	`BooleanGene`	true or false	Sampling flag for N_{n-1}

The genes in Part A differ according to the configuration parameters for a given learner. All the genes in Part B of a chromosome contain a Boolean allele. A chromosome's Part B has one gene for every majority instance in the dataset being tested. A gene value of true indicates that the corresponding data instance should remain in the training dataset, while a false value informs the system to remove that instance from the training data. Data sampling is only done from the training dataset, i.e., we never alter a fold's test dataset.

The chromosome definition for the GA-based experiments with MLP is shown in Table I. The numerical ranges shown in the table for the non-Boolean parameters are based on a combination of our prior machine learning experience and good coverage around the default parameter values as per WEKA [11]. If m represents the number of attributes in the given dataset, including the class attribute, then $\hat{m}$ is the lesser of m and 20. This is done from a computational efficiency consideration point of view.

The MLP chromosome allows for up to three hidden layers (`IntegerGene`). While two hidden layers are sufficient to approximate any continuous function [23], we allow elÀNN to consider up to three hidden layers with the aim of producing better results. However, given two MLP configurations with similar performances, elÀNN considers the less complex MLP as the better one. Part B of the chromosome has $N_0, N_1, \ldots, N_{n-1}$ genes, each representing a `BooleanGene` (true or false) for data sampling inclusion.

The fitness function of our GA system combines two effective performance factors, AUC and GMean. We opted to use both a threshold-independent factor (AUC) and a threshold-dependent factor (GMean) in an effort to evolve more generalized software quality classification models with less overfitting tendency [11]. The raw fitness, in its simplest form, of an individual X is given by the expression $f(X) = \alpha(AUC) + \beta(GMean)$, where α and β are constants which weigh the relative importance of the AUC and GMean factors, and add up to 1. We assigned a value of 0.5 to both α and β, making them equally important in the evolutionary process.

Given two individuals, we want to state that the better performer is fitter in every case. When two individuals perform similar we look at the sample size to decide which of the two is fitter – the one with a smaller sample size is better. The final fitness function for our data

sampling approach with MLP is given by,

$$f(X) = round\left(10^s\left[\alpha(AUC) + \beta(GMean)\right]\right) + \left(1 - \gamma\frac{k_H + k_O}{max(k_H + k_O)} - \delta\frac{I_N}{max(I_N)}\right) \quad (1)$$

In the above equation, s is the number of digits after the decimal place which we want to consider from the combined AUC-GMean raw fitness value – we opt for $s = 4$. The numbers of hidden and output nodes in the MLP network are given by k_H and k_O, respectively. We define complexity for the MLP model as the ratio of the actual number of hidden and output nodes to the maximum possible number of those nodes. We use ratios for both the network complexity and the sample size (I_N) so that they both respectively range between 0 and 1. γ and δ are weights used to adjust the importance of the MLP complexity and the sample size factors, where $\gamma + \delta = 1.0$. We consider both factors equally and assign a value of 0.5 to each.

The weighted sum of the network complexity and sample size factors will always range from (0,1]; hence, we subtract that sum from 1 because we want the function to minimize it as GA works to maximize the overall fitness. As we focus on binary classifiers, the MLP model will always have two output nodes ($k_O = 2$) as per WEKA [11]. k_H will vary according to the parameters encoded in Part A of an individual's chromosome when optimization of the MLP configuration is considered during evolution. When optimization of MLP configuration is not considered, we use a default value of three nodes in one hidden layer; hence, the complexity term equals one.

The following steps summarize the GA implemented in elANN and used for our empirical investigations. The goal of our GA system is to maximize the fitness function defined in Equation (1).

1) Create a population with the specified number of individuals (100 for the experiments in this research). Randomize the allele values for each individual's genes.
2) Create a sampling reference dataset with copies of the majority class instances. The order of the instances in this reference dataset matches the order of the genes used for evolutionary sampling in the individual.
3) Add a sampling index value to all the instance objects in the training data. The sampling index is simply the index of the instance in the reference dataset created in Step 2. It corresponds to the position of the evolutionary sampling gene in each individual's chromosome. elANN performs this step as a preprocessing measure to create a look-up system between the evolutionary sampling genes in the chromosome and the actual training data instances so that there will be no need to conduct searches for instances in the training data as the genetic algorithm progresses.
4) Create training and test pairs of datasets for each of the folds for five-fold cross-validation. Use the experiment number (1–20) to seed the random number generator for splitting the data. A five-fold cross-validation involves randomly dividing the software measurement dataset into five subsets, and training the classifier with four subsets while the remaining subset is used for testing (evaluating) the classifier. This process of training and testing is repeated five times, each time with a different evaluation dataset.
5) Loop for the desired number of generations:
    a) Run the "best chromosome" natural selector. Keep the fittest 75% of the population to take part in this generation and discard the remaining individuals.
    b) Apply the crossover operator. The number of crossovers to perform is equal to 50% of the current population size. For each mating in single-point crossover we randomly select two individuals from the population with replacement. The two new individuals are then added to the population to be processed by natural selection at the end of the generation.
    c) Apply the mutation operator. Each gene in all of the chromosomes in the population has a 25% chance of mutation. Mutation does not change the individual being mutated, but rather if an individual has one or more genes selected for mutation, the algorithm creates a copy of the individual and mutates the copy's genes. A mutation rate of 0.25 reflects the results of Tate [20] and Haupt [24], which suggest that for complex gene encodings (such as our GA system) a higher mutation value (instead of the traditional 0.1) would be more beneficial. Tate [20] also recommends increasing the mutation rate when one is unable to increase the population size, generally because of the computational overhead involved.
    d) Run the tournament natural selector to select individuals who will survive to the next generation. In each tournament of four randomly selected individuals, one is chosen for survival. The fittest individuals are more likely to be chosen, but it is not guaranteed that they will be.

In the algorithm above, an individual's fitness value is calculated on demand. When elANN calls for an individual's fitness for the first time, it performs the following steps:

1) Create five threads, one to handle each of the five folds for our cross-validation procedure. Each thread receives a copy of the fold training dataset and the fold test dataset created in Step 4 of the GA algorithm.
2) For each thread:
    a) For each instance in the fold training dataset, use the sampling index value stored with the instance in Step 3 of the GA algorithm to find the corresponding gene in the individual's chromosome. If the Boolean allele value for that gene is false, remove the instance from the thread's training dataset.
    b) Instantiate the selected WEKA classifier. This involves using the default parameters for the classifier if elANN is set for data sampling without classifier parameters' optimization.
    c) Build the classifier using the fold training dataset.
    d) Evaluate the classifier using the fold test dataset. Store the resulting performance metrics (AUC and GMean) for the individual.
3) Take an average of the performance metrics for the five folds to be used for calculating the individual's fitness. We use this average only to score the individual for the genetic algorithm. The GA considers the individual to be a composite of the five folds; however, we report all five scores for the cross-validation folds separately for the fittest individual at the end of the experiment. Thus, one experiment represents one model run on five (cross-validation) datasets and, therefore, gives us results for five runs.

We run each GA experiment 20 times, each time with a different seed for the pseudo-random number generator used for cross-validation as explained in Step 4 of the GA algorithm. At the end

of a set of experiments, we have results for 100 GA runs for one experiment. The average values of the respective performance metrics are reported.

V. Empirical Investigation

A. Case Study Datasets

The *CM1* project is a NASA mission software responsible for the monitoring and analysis of science instruments. Written in C, this project's data is made available through the Metrics Data Program at NASA. Each program module in the dataset provides the software metrics for a module or a function in the *CM1* project. The dataset includes 13 numeric and two nominal independent attributes and a nominal class attribute. The minority class consists of the software modules considered to be *fp*, while the majority class represents the modules which were *nfp*. There are a total of 505 modules, of which 48 (9.50%) are *fp* while 457 (90.50%) are *nfp*.

The *SP2* project contains software metrics and defect data for a release of a large-scale telecommunications system written in Protel, a language similar to Pascal [25]. Program modules in the dataset represent the source code modules in the software project. The 42 numeric independent attributes describe the program module's values for various software metrics. The dependent attribute is nominal and identifies the module as *fp* or *nfp*. The *fp* modules form the minority class, while the *nfp* modules form the majority class. There are a total of 3981 modules in *SP2*, of which 189 (4.75%) are *fp* while 3792 (95.25%) are *nfp*.

B. Experimental Settings, Results and Analysis

The evolutionary experiments were conducted with a population size of 100 individuals and for 100 generations. These specific values were based on considering a combination of computational practicality and obtaining relatively good results. However, it is likely that a larger population size and a longer evolution would yield better performances. An evolutionary experiment in our study involved 20 runs of stratified, five-fold cross-validations to train and test an MLP classifier with a given dataset.

We use the experiment run number (1-20) to seed the random number generator when creating the five cross-validation folds. A given run involves five instantiations of the classifier, where each involves one-fifth of the modules in the dataset being reserved for evaluating (testing) the classification model, while the remaining four-fifths are used to train the model. At the end, we have 100 values for our performance metrics, i.e., product of 20 runs and 5 folds. For a given GA experiment, this translates to 200 evolutionary computing models for the two datasets in our study. The two GA-based experiments (evolutionary sampling with, and without, MLP configuration optimization) develop a total of 400 evolutionary models. Considerable time and computing resources were involved to complete all the experiments in our study. This is a common observation in a typical evolutionary computing-based analysis.

The performance metrics for the MLP models trained both without any data sampling technique and with Evolutionary Sampling are shown in Table II. The *BAS* column represents MLP models trained without any data sampling technique and with default options for the MLP learner of WEKA. The *EVS-DF* column represents MLP models trained with default parameter configurations (of WEKA) and with Evolutionary Sampling. The *EVS-OT* column represents MLP models trained with both Evolutionary Sampling and with optimization/tuning of the MLP parameter configurations. The MLP models are evaluated on the F-Meas, AUC, and GMean performance metrics. Recall, F-Meas was not used in our fitness function as compared to the AUC and GMean.

TABLE II
Results of GA Experiments With MLP

F-Meas performance			
Dataset	*BAS*	*EVS-DF*	*EVS-OT*
CM1	0.053	0.431	0.415
SP2	0.129	0.327	0.342
AUC performance			
Dataset	*BAS*	*EVS-DF*	*EVS-OT*
CM1	0.812	0.840	0.838
SP2	0.829	0.841	0.845
GMean performance			
Dataset	*BAS*	*EVS-DF*	*EVS-OT*
CM1	0.094	0.759	0.754
SP2	0.260	0.608	0.639

TABLE III
Majority Undersampling Techniques' Results

F-Meas performance				
Dataset	*EVS-DF*	*RUS*	*WLE*	*OSS*
CM1	0.431	0.312	0.198	0.094
SP2	0.327	0.272	0.166	0.147
AUC performance				
Dataset	*EVS-DF*	*RUS*	*WLE*	*OSS*
CM1	0.840	0.807	0.805	0.809
SP2	0.841	0.834	0.832	0.830
GMean performance				
Dataset	*EVS-DF*	*RUS*	*WLE*	*OSS*
CM1	0.759	0.676	0.332	0.162
SP2	0.608	0.714	0.313	0.286

The *EVS-DF* and *EVS-OT* models clearly outperform the *BAS* models, making a clear case for addressing the rarity of fault-prone modules in software quality modeling. This is true for both software measurement datasets and for all three performance metrics. An ANOVA and Tukey's Honestly Significant Difference (HSD) statistical analysis indicated that the *EVS-DF* and *EVS-OT* models are significantly better than the *BAS* models at $\alpha = 0.05$ [10]. A comparison between the *EVS-DF* and *EVS-OT* models indicates that the two modeling approaches are relatively competitive with respect to all three performance metrics.

We compare our EVS approach with other existing majority undersampling techniques, and those results are summarized in Table III. The table reflects classifier performances of MLP learners trained using the default parameter configuration in WEKA. The table does not include the *EVS-OT* results because those software quality models were trained with optimization of modeling parameters for the MLP learner, as discussed earlier. The *RUS* results are the best performances among random undersampling rates of 5%, 10%, 25%, 50%, 75%, and 90% for a given dataset. The results for *WLE* represent the better value of the standard and the weighted versions [8].

In the case of the smaller software measurement dataset, *CM1*, the *EVS-DF* models are always better than all of the other majority undersampling techniques. This is true for all three performance metrics considered in our study. In the case of the larger software measurement dataset, *SP2*, the *EVS-DF* models have the best performances with respect to F-Meas and AUC. However, with GMean as the performance metric, the *RUS* technique gave the best results. An

ANOVA and Tukey's HSD statistical evaluation revealed that when combining all results from both datasets, the *EVS-DF* models were better than the other majority undersampling techniques [10].

VI. Conclusion

Evolutionary Sampling is presented as a viable data sampling technique for addressing rarity of the minority class instances in a machine learning dataset. This paper addresses the problem in the context of software quality classification modeling where the proportion of fault-prone modules is often a very small fraction of the software measurement training dataset.

Implemented as a research prototype, eLǸNN, the proposed majority undersampling technique can also be combined with a genetic algorithm-based optimization of the different modeling parameters of a machine learner. This paper focuses on using the multilayer perceptron neural network as the binary classifier of choice; however, the proposed approach and comparative study can be extended to most other binary classifiers. Currently eLǸNN implements the C4.5 decision tree, RIPPER, and Multilayer Perceptron learners, and is based on the WEKA data mining tool framework.

Software measurement datasets from two real-world software projects are used as case studies for evaluating the proposed Evolutionary Sampling approach. In addition to comparing the before and after Evolutionary Sampling software quality models, this study also compares EVS with three other majority undersampling techniques. They are random undersampling, Wilson's editing, and one-sided selection. A comparison with other data sampling techniques, i.e. minority oversampling, is not presented due to space limitations.

Empirical results of the case studies presented clearly indicate that EVS improves the software quality model's performance as compared to modeling without any data sampling technique. A classifier is evaluated based on three independent performance metrics, namely Area-Under the ROC Curve, Geometric Mean, and F-Measure. Compared to the other majority undersampling techniques, the proposed approach shows a significant improvement, especially when F-Meas and AUC are considered as performance metrics. Among the two software measurement datasets, random undersampling fares competitively for the larger dataset. This is likely due to the availability of good majority class instances even after randomly eliminating some majority class instances.

Some directions for future work include further validation with additional datasets and optimizing the different GA parameters such as population size, number of generations, mutation rate, and crossover rate. While GA parameters' optimization would likely improve the end results, such analysis comes at the expense of increased computational and time complexities. However, some of that concern can be alleviated since GA generally lends itself well to parallelism, and parallel systems can provide added benefits to using evolutionary techniques.

Acknowledgment

We thank the various members of the Empirical Software Engineering Laboratory and the Data Mining and Machine Learning Laboratory of Florida Atlantic University for assistance with reviews.

References

[1] L. Guo, B. Cukic, and H. Singh, "Predicting fault prone modules by the dempster-shafer belief networks," in *Proceedings of the 18th International Conference on Automated Software Engineering*. Montreal, Quebec, Canada: IEEE Computer Society, October 2003, pp. 249–252.

[2] T. M. Khoshgoftaar and N. Seliya, "Comparative assessment of software quality classification techniques: An empirical case study," *Empirical Software Engineering Journal*, vol. 9, no. 3, pp. 229–257, 2004.

[3] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, Jan. 2007.

[4] K. S. Woods, C. C. Doss, K. W. Bowyer, J. L.Solka, C. E. Priebe, and W. P. Kegelmeyer, "Comparative evaluation of pattern recognition techniques for detection of microcalcifications in mammography," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 6, pp. 1417–1436, 1993.

[5] M. Kubat, R. C. Holte, and S. Matwin, "Machine learning for the detection of oil spills in satellite radar images," *Machine Learning*, vol. 30, no. 2-3, pp. 195–215, 1998. [Online]. Available: citeseer.ist.psu.edu/article/kubat98machine.html

[6] R. Lippmann, J. Haines, D. Fried, J. Korba, and K. Das, "The 1999 darpa off-line intrusion detection evaluation," *Computer Networks*, vol. 34, no. 4, pp. 579–595, 2000.

[7] G. M. Weiss, "Mining with rarity: A unifying framework," *SIGKDD Explorations*, vol. 6, no. 1, pp. 7–19, 2004.

[8] R. Barandela, R. M. Valdovinos, J. S. Sánchez, and F. J. Ferri, "The imbalanced training sample problem: Under or over sampling?" in *Syntatical and Structural Pattern Recognition/Statistical Pattern Recognition*, 2004, pp. 806–814.

[9] M. Kubat and S. Matwin, "Addressing the curse of imbalanced training sets: One sided selection," in *Proceedings of the Fourteenth International Conference on Machine Learning*. Morgan Kaufmann, 1997, pp. 179–186.

[10] D. J. Drown, "Evolutionary methods for mining data with class imbalance," Master's thesis, Florida Atlantic University, Dept. of Computer Science and Engineering, Boca Raton, Florida, USA, August 2007, advised by Taghi M. Khoshgoftaar.

[11] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd ed. San Francisco, California: Morgan Kaufmann, 2005.

[12] D. Wilson, "Asymptotic properties of nearest neighbor rules using edited data sets," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 2, pp. 408–421, 1972.

[13] I. Tomek, "Two modifications of CNN," *IEEE Transactions on Systems Man and Communications SMC-6*, pp. 769–772, 1976.

[14] A. Orriols and E. Bernadó-Mansilla, "Class imbalance problem in UCS classifier system: fitness adaptation," in *IEEE Congress on Evolutionary Computation*, September 2005, pp. 604–611.

[15] F. Provost and T. Fawcett, "Robust classification for imprecise environments," *Machine Learning*, vol. 42, pp. 203–231, 2001.

[16] A. E. Napolitano, "Alleviating class imbalance using data sampling: Examining the effects on classification algorithms," *Master's Thesis, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, Florida, USA*, December 2006, advised by Taghi M. Khoshgoftaar.

[17] R. Hochman, "Software reliability engineering: An evolutionary neural network approach," *Master's Thesis, College of Engineering, Florida Atlantic University, Boca Raton, Florida, USA*, 1997, advised by Taghi M. Khoshgoftaar.

[18] A. A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Heidelberg, Germany: Springer-Verlag, 2002.

[19] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*. Norwell, Massachusetts: Kluwer Academic Publishers, 2001.

[20] D. M. Tate and A. E. Smith, "Expected allele coverage and the role of mutation in genetic algorithms," in *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, S. Forrest, Ed. San Mateo, CA: Morgan Kaufmann, 1993, pp. 31–37.

[21] T. Jones, *AI Application Programming*, 2nd ed. Hingham, MA: Charles River Media, 2005.

[22] K. Meffert and N. Rotstan, "JGAP: Java genetic algorithms package," jgap.sourceforge.net, 2007.

[23] J. Principe, N. Euliano, and W. C. Lefebvre, *Neural and Adaptive Systems: Fundamentals through Simulation*. New York: John Wiley & Sons, 2000.

[24] R. L. Haupt, "Adaptive crossed dipole antennas using a genetic algorithm," *IEEE Transactions on Antennas and Propagation*, vol. 52, no. 8, pp. 1976–1982, 2004.

[25] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl, "Classification-tree models of software-quality over multiple releases," *IEEE Trans. Reliability*, vol. 49, no. 1, pp. 4–11, March 2000.

Machine Learning and Value-Based Software Engineering: A Research Agenda

Du Zhang
Department of Computer Science
California State University
Sacramento, CA 95819-6021
zhangd@ecs.csus.edu

Abstract

Software engineering research and practice thus far are primarily conducted in a value-neutral setting where each artifact in software development such as requirement, use case, test case, and defect, is treated as equally important during a software system development process. There are a number of shortcomings of such value-neutral software engineering. Value-based software engineering is to integrate value considerations into the full range of existing and emerging software engineering principles and practices.

Machine learning has been playing an increasingly important role in helping develop and maintain large and complex software systems. However, machine learning applications to software engineering have been largely confined to the value-neutral software engineering setting. In this paper, the general message to be conveyed is to apply machine learning methods and algorithms to value-based software engineering. The training data or the background knowledge or domain theory or heuristics or bias used by machine learning methods in generating target models or functions should be aligned with stakeholders' value propositions. An initial research agenda is proposed for machine learning in value-based software engineering.

Keywords: *value-based software engineering, machine learning, stakeholder value propositions, Pareto modules*.

1. Introduction

Software engineering research and practice thus far are mainly conducted in a value-neutral setting where each artifact in software development such as a requirement, a use case, a test case, a defect, and so forth, is treated as equally important during a software system development process [2]. There are a number of shortcomings of such value-neutral software engineering [1]: (1) its exclusion of economics, management sciences, cognitive sciences, and humanities from the body of knowledge needed to develop successful software systems; (2) its delimitation of software development by mere technical activities; and (3) its failure to explicitly recognize the fact that software systems continue to satisfy and conform to evolving human and organizational needs is to create value. Value-based software engineering (VBSE) is to integrate value considerations into the full range of existing and emerging software engineering principles and practices so as to increase the return on investment (ROI = (benefits–costs)/costs) for the stakeholders and optimize other relevant value objectives of software projects [1, 2].

Machine learning (ML) has been playing an increasingly important role in helping develop and maintain large and complex software systems. However, machine learning applications to software engineering have been largely confined to the value-neutral software engineering setting [29-31,33]. In this paper, the general message we attempt to convey is to apply ML methods beyond the value-neutral software engineering setting and to VBSE. The training data or the background knowledge or domain theory or heuristics or bias used by ML methods in generating target models or functions for software development and maintenance should be aligned with stakeholders' value propositions (SVPs) and business objectives. Even though the transition to VBSE from the traditional value-neutral setting is necessarily evolutionary because not all the theories, infrastructures, methodologies and tools for VBSE have been fully developed yet, there are a number of agenda items for VBSE [2].

The goal of the road map in VBSE is to make software development and maintenance decisions that are better for value creation [2]. On the other hand, the hallmark of ML is that it results in an improved ability to make better decisions. VBSE offers a fertile ground where many software development and maintenance tasks can be formulated as ML problems and approached in terms of ML methods. The purpose of this paper is to describe an initial research agenda for ML applications to VBSE with regard to the identified areas in VBSE (value-based requirement engineering, architecting, design and development, verification and validation, planning and control, risk/quality/people managements, and a theory of VBSE [2]).

The rest of the paper is organized as follows. Section 2 offers an overview of the related work. Section 3 highlights some important concepts in VBSE. In Section 4, we describe an initial research agenda for ML applications in VBSE. Finally Section 5 concludes the paper with remark on future work.

2. Related Work

In addition to machine learning in (value-neutral) software engineering (MLSE), there are a number of related and emerging software development paradigms: search-based software engineering (SBSE), evidence-based software engineering (EBSE), model-based software engineering (MBSE), artificial intelligence in software engineering (AISE), and computational intelligence in software engineering (CISE). We provide a brief account for each of the paradigm. Figure 1 highlights their similarities and differences.

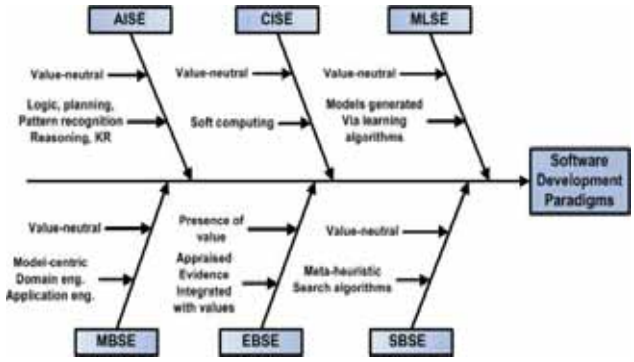


Figure 1. Emerging software development paradigms.

2.1. MLSE

ML falls into the following broad categories: *supervised* learning, *unsupervised* learning, *semi-supervised* learning, *analytical* learning, *reinforcement* learning, and *multi-agent* learning. Each of the categories in turn includes various learning methods. Supervised learning deals with learning a target function from labeled examples. Unsupervised learning attempts to learn patterns and associations from a set of objects that do not have attached class labels. Semi-supervised learning is learning from a combination of labeled and unlabeled examples. Analytical learning relies on domain theory or background knowledge to learn a target function. Reinforcement learning is concerned with learning a control policy through reinforcement from an environment. Multi-agent learning is an extension to single-agent leaning. There are many emerging learning methods such as argument based machine learning, interactive learning, transfer learning, and so forth.

In software development, there are processes, products and resources [9], which in turn have internal and external attributes. Internal attributes describe an entity itself, whereas external attributes characterize the behavior of an entity (how the entity relates to its environment).

A partial list of ML applications in value-neutral software engineering includes [29-31, 33]: (1) Predicting or estimating measurements for either internal or external attributes of software development processes, products, or resources. (2) Discovering either internal or external properties of the processes, products, or resources. (3) Transforming products to accomplish some desirable or improved external attributes. (4) Synthesizing or generating various products. (5) Reusing products or processes. (6) Enhancing processes. (7) Managing products.

There were many different ML methods utilized in the aforementioned applications [29-31, 33]. A common property in the existing ML applications is that the software engineering issues were tackled solely from technical or logical perspectives (involving mappings and transformations, for instance) without the value dimension being taken into consideration (e.g., how to increase ROI for the stakeholders and optimize other relevant value objectives of software projects). The training data or the background knowledge or domain theory or heuristics or bias used by the ML methods in generating target functions did not contain any value propositions.

2.2. SBSE

SBSE treats software development tasks as a search problem with regard to a set of constraints and a search space of possible solutions [6, 12]. It relies on evolutionary algorithms, gradient ascent/descent, particle swarm intelligence, simulated annealing, tabu search or colony optimization techniques to tackle the software development or maintenance tasks. So far its applications have included the following areas in software engineering: requirement engineering, project planning, cost estimation, maintenance, reverse engineering, refactoring, program comprehension, service oriented tasks, quality assessment, and testing (structural, functional, non-functional, state-based properties, robustness, stress, security, mutation, regression, interaction, integration, and exception). Value considerations are not explicitly incorporated into the search process.

2.3. EBSE

EBSE is geared toward improving the decision making process related to software development and maintenance by integrating current best evidence from research with practical experience and human values [7, 15]. There are five main steps in EBSE as delineated in [7, 15]: (1) Translate a relevant problem or need of information into an answerable question. (2) Glean the literature for the best available evidence that can be used to answer the question. (3) Assess the evidence for its validity, impact, and applicability. (4) Combine the appraised evidence with practical experience, and stakeholders' values and circumstances to make decisions. (5) Evaluate performance and find ways to improve it. An important strength of EBSE is that it does take into consideration SVPs.

2.4. MBSE

MBSE is centered on software models, modeling, and model transformation technologies. It is a disciplined approach to developing and extending a product family. The software models provide the necessary information to support, economically and effectively, future changes to a software product family. By focusing on models that capture and consolidate developers' understanding of a family of software products, reusable assets can be developed that satisfy a wide variety of uses and can be utilized to analyze existing software to quickly compose or synthesize new solutions for subsequent products in a product family [4, 21, 26]. The goal is to achieve the benefits of reuse, shorter time to market, product maintainability and higher quality. However, value considerations are not prominently factored into the paradigm.

MBSE consists of two parallel engineering processes: domain engineering and application engineering, and sanctions the concepts of product families, a production system, and software assets (the reusable resources needed in application engineering such as domain models, software architectures, design standards, communication protocols, code components and application generators).

Many organizations have model-based development paradigm in place: Microsoft's Software Factory [10], Lockheed Martin's Model Centric Software Development [28], and NASA JPL's Defect Detection and Prevention [8].

2.5. CISE

In CISE (or software engineering with computational intelligence), soft computing techniques such as fuzzy sets, neural networks, genetic algorithms, genetic programming and rough sets (or combinations of those individual technologies) are utilized to tackle software development issues recently [13, 14, 16, 17, 23]. The results have been largely confined to the value-neutral setting.

2.6. AISE

The application of some general artificial intelligence techniques to software engineering (AISE) has produced some encouraging results [19, 20, 22, 25, 27]. Some of the successful AI techniques include: knowledge-based approach, automated reasoning, expert systems, heuristic search strategies, temporal logic, planning, and pattern recognition. Again the results thus far have been obtained in the value-neutral setting.

3. VBSE

The essence in VBSE is that the approach aims at aligning software development and maintenance with customer requirements and strategic business objectives. It offers a framework where SVPs are incorporated into the technical and managerial decisions made during software development and maintenance [1, 11].

Value includes product, process and resource attributes. Value attributes include: profits (generated from products), strategic positioning in market share, utility, relative worth, reputation, customer loyalty, innovation technology, cost reduction, quality of life, improved productivity.

An emerging agenda of issues in VBSE has been proposed in [2], that includes the following areas:

- Value-based requirements engineering. The key objectives include recognition of success-critical stakeholders, elicitation of SVPs, and reconciliation of SVPs.
- Value-based architecturing. The goals are to iron out the discrepancy between a system's objectives and achievable architectural solutions.
- Value-based design and development. The goals are to ensure that a software system's objectives and its value considerations are embodied in the software's design and development practices.
- Value-based verification and validation. The objectives are to ascertain that a software solution meets its value objectives and that V&V tasks are sequenced and prioritized as investing activities.
- Value-based planning and control. The objectives in this area are to incorporate the value delivered to stakeholders into the product planning and control techniques.
- Value-based risk management. How to factor the value considerations into principles and practices for risk identification, analysis, prioritization, and mitigation is the main focus in this area.
- Value-based quality management. The goals are to prioritize desired software quality considerations with respect to SVPs.
- Value-based people management. The tasks involve building stakeholder team, manage expectations, and reconcile SVPs.

To facilitate ML applications in VBSE, a number of concepts need to be in place, one of which is about Pareto modules.

Figure 2 depicts a reported study in [5] where the dotted line reflects the value-neutral practice in which an automated test data generation tool assumes that all tests have the same value. The Pareto curve for the empirical data, on the other hand, displays the actual business value where one of the fifteen customer services accounted for 50% of all billing revenues.

We refer to module(s) that realizes a service of such a high positive impact on the system's ROI as *Pareto modules*. They are the most important modules of a software system with regard to its product value. How

modules contribute to a product's overall value hinges on reconciled SVPs.

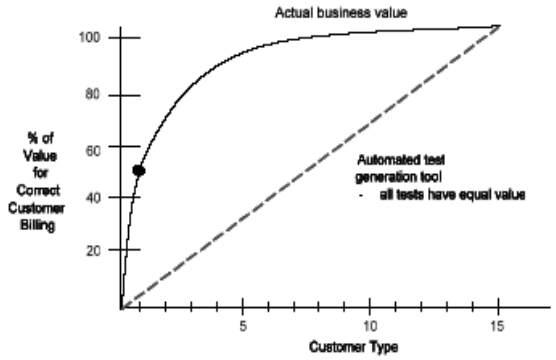


Figure 2. Pareto distribution for varying test case value.

For a given software system Ω, we can define a set $\mathbf{M}_\Omega$ of modules, a valuation function υ, and a value set $\boldsymbol{V}$ as follows:

$\mathbf{M}_\Omega = \{m_i \mid m_i \in \Omega\}$;
$\upsilon: \mathbf{M}_\Omega \rightarrow [0, 1]$;
$\boldsymbol{V} = \{\upsilon(m_i) \mid m_i \in \mathbf{M}_\Omega\}$

The valuation function υ can be defined by SVPs and has the following properties:

- $0 < \upsilon(m_i) \leq 1$, for all i;
- $\sum_{i=1}^{n} \upsilon(m_i) = 1$.

We define a partially ordered set $(\boldsymbol{V}, \leq)$ where $\leq$ is a binary order relation on $\boldsymbol{V}$ and satisfies reflexivity, anti-symmetry and transitivity for all elements in $\boldsymbol{V}$. We say that $\upsilon(m_p)$ is a *principal element* for $(\boldsymbol{V}, \leq)$ if we have the following:

$$\forall \upsilon(m_j) \in \boldsymbol{V}\ [\upsilon(m_j) \leq \upsilon(m_p)]^1$$

We use ρ to denote the *principal module* as specified by $\upsilon(m_p)$. We define a *principal-element-ordered subset* $\boldsymbol{V}_{[m_i, \rho]}$ of $\boldsymbol{V}$ and its cumulative value $\mu_{[m_i, \rho]}$ as follows:

$$\boldsymbol{V}_{[m_i, \rho]} = \boldsymbol{V} - \{\upsilon(m_k) \mid \upsilon(m_k) \leq \upsilon(m_i)\}$$
$$\mu_{[m_i, \rho]} = \sum \upsilon(m_j) \in \boldsymbol{V}_{[m_{i,} \rho]}$$

Now we are in a position to formally define the concept of Pareto modules.

Definition 1. Given a threshold value $\tau \in (0, 1]$, we identify a principal-element-ordered subset $\boldsymbol{V}_{[m_i, \rho]}$ such that $\tau = \mu_{[m_i, \rho]}$. Modules in $\boldsymbol{V}_{[m_i, \rho]}$ are referred to as Pareto modules with regard to τ.

If $\tau < \mu_{[m_i, \rho]}$ but removing any m_j from $\mu_{[m_i, \rho]}$ would result in $\tau > \mu_{[m_i, \rho]}$, then the condition of $\tau = \mu_{[m_i, \rho]}$ is relaxed to that of $\tau \leq \mu_{[m_i, \rho]}$.

[1] If there are several principal elements in $\boldsymbol{V}$, we can use other criteria to designate one for the discussion.

4. Research Agenda for ML in VBSE

In this section, we first discuss some general issues on how to calibrate ML methods for VBSE tasks. Using Boehm' VBSE agenda in [2] as a roadmap, we then describe some preliminary agenda items of how ML can help with the goals, objectives and tasks in VBSE.

4.1. How to calibrate ML methods

ML methods formulate various general hypotheses, models and target functions through either observed training data, or some background knowledge or domain theory, or a combination of both. The generalization process during leaning also hinges on certain adopted bias or heuristics.

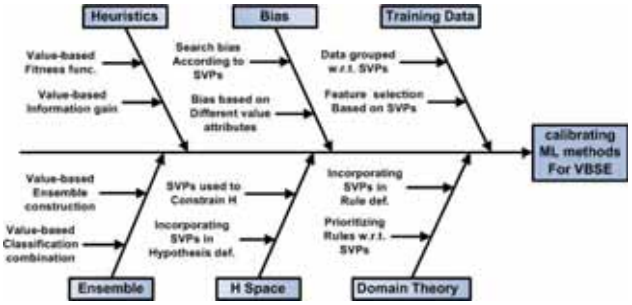


Figure 3. Calibrating ML methods for VBSE.

To calibrate ML methods for VBSE tasks, the fundamental issue is how to incorporate SVPs from the business value level into the technical level details of ML model generation process. Specifically, this translates into the following issues: how to use SVPs to select data features and to group training data, how to incorporate SVPs into domain knowledge representation, how to prioritize rules, based on SVPs, in domain knowledge during model generation, how to include SVPs in defining search bias, how to use different value attributes in defining domain-specific biases for the search process, how to utilize SVPs in defining hypotheses and constraining hypothesis space, how to factor SVPs into ensemble construction and classification combination process when ensemble learning is used to generate models, and how the value concept plays a role in defining ML method-specific heuristics (e.g., fitness function, information gain measure).

4.2. Value-based requirements engineering

For the objectives in value-based requirements engineering, techniques such as business case analysis, requirements prioritization and release prioritization have proven to be effective [2].

ML methods can be utilized to assist business case analysis, and requirements and release prioritization. Specifically, ML methods can be used to predict or estimate software cost, software size, software development efforts, and release prioritization and timing.

These prediction, estimation or cost models would help stakeholders gain insight on what capabilities are not feasible with regard to budget, schedule and technology constraints, which features of a system are most important and attainable, and which aggregate of capabilities will meet stakeholders' critical needs given the resource constraints. This in turn will assist stakeholders in prioritizing and reconciling potential conflicting value propositions. Possible ML methods for generating the models include: decision trees, Bayesian learning, neural networks, genetic algorithms, genetic programming, case-based reasoning, and inductive logic programming.

4.3. Value-based verification and validation

The key techniques in value-based V&V are value-based and risk-based testing techniques [2]. Central to those techniques is how to align SVPs with the technical level details in test construction and execution and how to sequence and prioritize testing activities as investing activities [24, 32].

Since we only have the Pareto module concept in place, we need to introduce defect-intensive and defect-prone modules, impact and non-impact defects before we can use an SVPs-based approach to identify modules of the aforementioned types and decompose the overall test data generation process for a given software system into a sequence of test data generation cycles with each focusing on some specific testing objective(s). From a value-based standpoint, to improve the return on investment, we want to make sure to first maximize the success rate of Pareto modules and to minimize the chance of having impact defects that devastate the value contribution. Afterwards, attention can be focused on non-impact defects and non-Pareto modules. Thus, a prioritization of cycles can be generated that is driven by the value consideration and allows the most critical modules with regard to SVPs to be thoroughly tested first.

For each cycle, a number of ML methods can be utilized to generate test cases for different classes of modules. Some possible ML methods include: genetic algorithms [18], genetic programming, inductive logic programming and rule-based active learning.

4.4. Value-based risk management

There are a number of techniques for value-based risk management: the risk-based "how-much-is-enough" techniques, the risk-based analysis for project predictability, the risk-based simulation, and the risk-based testing techniques [2].

A pivotal concept in risk management is the *risk exposure* (***RE***) involved in a prescribed course of actions. ***RE*** is defined as follows:

$$\boldsymbol{RE} = \boldsymbol{P}(\mathscr{L}) \times \boldsymbol{S}(\mathscr{L})$$

where $\boldsymbol{P}(\mathscr{L})$ is the probability of loss $\mathscr{L}$, and $\boldsymbol{S}(\mathscr{L})$ is the size of loss. $\mathscr{L}$ can be defined based on any value attribute as discussed in Section 3. In the risk exposure profile analysis [3], there is a dichotomy between planning and market share as the value attribute: inadequate planning results in little delay to capture market share but high ***RE*** due to oversights and rework; excessive planning reduces the chance of major problems but at the expense of high ***RE*** because of time-to-market delays.

ML methods can be used to help find the "sweet spot" for different risk profiles and different risk exposure profiles. Depending on the circumstances, either inductive learning, or analytical learning, or a combination of inductive and analytical learning can be deployed.

4.5. Value-based design and development

To ensure that a system's objectives and its value considerations are embodied in the software's design and development practices, the software traceability techniques play an important role [2]. During the software development process, many artifacts are produced and maintained: documents, requirements, design models, test scenarios, and so forth. Trace dependencies are to identify relationships among those artifacts and the quality of the trace dependencies should reflect the value of the artifacts they attempt to bridge. This is vital for a number of reasons, from documentation, program understanding, impact analysis, consistency checking, reuse, quality assurance, user acceptance, error reduction, cost estimation, to customer satisfaction.

ML methods can be used to establish value-based trace dependencies among different artifacts. Methods such as instance-based learning (case-based reasoning), inductive logic programming, rule-based learning would lend themselves to the task.

4.6. Value-based quality management

ML methods can be used to generate predictive models for identifying high risk or fault prone components as an integral part of the quality management. Because of the need to align desired quality properties with SVPs, value considerations should be, directly or indirectly, involved in defining or contributing to those quality properties. SVPs should also help prioritize the desired quality factors.

ML methods that are appropriate for the task include: decision trees, genetic programming, neural networks, case-based reasoning, inductive logic programming, and concept learning.

5. Conclusion

VBSE offers a new software development paradigm that recognizes the importance of business and customer value considerations. It tackles the decision making process in software development and maintenance from a value-based perspective. In this paper, we discuss the issue of ML applications to VBSE. Because ML applications to software engineering thus far have been

largely confined to the value-neutral setting, we reviewed the landscape in the field and took a closer look at the emerging agenda for VBSE to find out how ML can be positioned to play a larger role in VBSE. We propose some guideline on how to calibrate ML methods to accommodate the value considerations that are so critical in accomplishing VBSE agenda items. Using Boehm' VBSE roadmap as a guide, we describe some preliminary agenda items on how ML can help with the goals, objectives and tasks in VBSE.

The take-home message of this work is two-fold: VBSE offers a ROI-conscious approach to software development and maintenance, and ML has an active and important role to play in various agenda items in VBSE.

The viability of ML applications in VBSE hinges on the outcomes of empirical studies, which will be pursued as our future work. How to solidify SVPs into various ML methods is an open issue worth studying.

References

1. S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grunbacher (ed.), *Value-Based Software Engineering*, Springer, Berlin, 2006.
2. B. Boehm, "Value-Based Software Engineering: Overview and Agenda," in *Value-Based Software Engineering*, S. Biffl et al (ed.), Springer, Berlin, 2006.
3. B. Boehm, "Value-Based Software Engineering: Seven Key Elements and Ethical Considerations," in *Value-Based Software Engineering*, S. Biffl et al (ed.), Springer, Berlin, 2006.
4. A. Brown, S. Iyengar and S. Johnston, "A Rational Approach to Model-Based Development," *IBM Systems Journal*, Vol. 45, No. 3, 2006, pp.463-480.
5. J. Bullock, "Calculating the Value of Testing," *Software Testing and Quality Engineering*, May/June issue, 2000, pp.56-62.
6. J. Clark et al, "Reformulating Software Engineering as A Search Problem," *IEE Proceedings – Software*, Vol. 150, No. 3, 2003, pp.161-175.
7. T. Dyba, B. A. Kitchenham and M. Jorgensen, "Evidence-Based Software Engineering for Practitioners," *IEEE Software*, Vol. 22, No. 1, 2005, pp.58-65.
8. M. Feather et al, "A Broad, Quantitative Model for Making Early Requirements Decisions," *IEEE Software*, Vol. 25, No. 2, 2008, pp.49-56.
9. N. E. Fenton and S. L. Pfleeger, *Software Metrics*, PWS Publishing Company, 2nd ed., 1997.
10. J. Greenfield and K. Short, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley Publishing, Indianapolis, IN, 2004.
11. P. Grunbacher, S. Koszegi and S. Biffl, "Stakeholder Value Proposition Elicitation and Reconciliation," in *Value-Based Software Engineering*, S. Biffl et al (ed.), Springer, Berlin, 2006.
12. M. Harman and B. Jones, "Search-Based Software Engineering," *Information and Software Technology*, Vol. 43, No. 14, 2001, pp.833-839.
13. T. Khoshgoftaar, *Software Engineering with Computational Intelligence*, Kluwer, 2003.
14. T. Khoshgoftaar (ed.), Special Issue on Quality Engineering with Computational Intelligence, *Software Quality Journal*, Vol.11, No.2, June 2003.
15. B. A. Kitchenham, T. Dyba and M. Jorgensen, "Evidence-Based Software Engineering," in *Proceedings of International Conference on Software Engineering*, 2004, Edinburgh, pp.273-281.
16. J. Lee, *Software Engineering with Computational Intelligence*, Springer-Verlag, 2003.
17. J. Lee (ed.), Special Issue on Software Eng with Computational Intelligence, *Information and Software Technology*, Vol.45, No.7, May 2003.
18. P. McMinn, "Search-based Software Test Data Generation: A Survey," *Software: Testing, Verification and Reliability*, Vol. 14, No. 2, 2004, pp. 105-156.
19. M. Mendonca and N.L. Sunderhaft, "Mining software engineering data: a survey", DACS State-of-the-Art Report, September 1999, http://www.dacs.dtic.mil/techs/datamining/.
20. J. Mostow (ed), Special issue on artificial intelligence and software engineering, *IEEE Trans. SE*, Vol.11, No.11, November 1985, pp.1253-1408.
21. MBSE: http://www.sei.cmu.edu/mbse/index.html.
22. D. Partridge, *Artificial Intelligence and Software Engineering*, AMACOM, 1998.
23. W. Pedrycz and J.F. Peters, *Computational Intelligence in Software Engineering*, World Scientific Publisher, 1998.
24. R. Ramler, S. Biffl and P. Grunbacher, "Value-Based Management of Software Testing," in *Value-Based Software Engineering*, S. Biffl et al (ed.), Springer, Berlin, 2006.
25. C. Rich and R. Waters (eds.), *Readings in Artificial Intelligence and Software Engineering*, Morgan Kaufmann, 1986.
26. S. Sendall and W. Kozacaynski, "Model Transformation: the Heart and Soul of Model-Driven Software Development," *IEEE Software*, Vol. 20, No. 5, 2003, pp.42-45.
27. J.J.P. Tsai and T. Weigert, *Knowledge-Based Software Development for Real-Time Distributed Systems*, World Scientific Inc., Singapore, 1993.
28. D. Waddinggton and P. Lardieri, "Model-Centric Software Development," *IEEE Computer*, Vol. 39, No. 2, 2006, pp.28-29.
29. D. Zhang, "Applying Machine Learning Algorithms in Software Development," *Proceedings of Monterey Workshop on Modeling Software System Structures*, Santa Margherita Ligure, Italy, June 2000, pp.275-285.
30. D. Zhang and J. J. P. Tsai, "Machine Learning and Software Engineering," *Software Quality Journal*, Vol.11, No.2, June 2003, pp.87-119.
31. D. Zhang and J. J. P. Tsai (ed.), *Machine Learning Applications in Software Engineering*, World Scientific Publishing Co., Singapore, 2005.
32. D. Zhang, "Machine Learning in Value-Based Software Test Data Generation," the Proceedings of the *Eighteenth IEEE International Conference on Tools with AI,* Washington DC, November 2006, pp.732-736.
33. D. Zhang and J. J. P. Tsai (ed.), *Advances in Machine Learning Applications in Software Engineering*, Idea Group Publishing, Hershey, PA, 2007.

Automatic Clustering of Defect Reports

Vasile Rus
Department of Computer Science
The University of Memphis
373 Dunn Hall
Memphis, TN
vrus@memphis.edu

Sameer Mohammed
Department of Computer Science
The University of Memphis
373 Dunn Hall
Memphis, TN
smohmmd1@memphis.edu

Sajjan Shiva
Department of Computer Science
The University of Memphis
373 Dunn Hall
Memphis, TN
sshiva@memphis.edu

Abstract

This paper addresses the problem of clustering defect reports. Clustering defect reports can provide valuable information to software testers, e.g. it could help better plan and prioritize the testing effort as testers could focus on testing the features with most defects as indicated by the largest clusters identified. In this paper, we present results obtained with one clustering algorithm, K-means, and two models of defect reports. In one model we use the summary field of the reports and in another the description field. Our experiments on defect reports from Mozilla's Bugzilla, a database of defect reports related to the open source Mozilla project, showed that clustering defect reports based on their summary field (average accuracy=44.2240%) outperformed clustering based on the description field (average accuracy=29.3581%). Both methods outperform the baseline of randomly picking a cluster (accuracy=20.0000%). We evaluated the clustering algorithm with respect to clusters containing bug reports that refer to the same underlying problem.

1. Introduction

We address in this paper the challenging task of clustering defect reports. Defect reports are detailed descriptions in natural language of defects, i.e. problems in a software product. The quality and proper handling of defect reports throughout the testing process will have a great impact on the quality of the released software product. The defect reports are currently analyzed manually by testers, developers, and other stakeholders. Manual analysis is tedious, error-prone, and time consuming, leading to a less efficient testing process.

We propose here automatic methods to analyze defect reports. In particular, we propose automatic methods to clustering defect reports in order to discover patterns among reported defects. Clustering could reveal sets of related problems and this information could be further used to better plan the testing effort. For instance, a large cluster of related reports could indicate which feature(s) of the software product needs to be further tested. While a cluster of defect reports that look similar content-wise may not always describe the same underlying bug, i.e. root cause, they could highlight visible features of a product that need more attention from the testing team. However, in this paper we address the more challenging task of clustering defect reports based on their describing the same underlying bug. This is possible by evaluating the clustering using reports that were judged by developers as being duplicates, i.e. describing the same bug.

Defect reports are filed by testers (or users) who discover the defects through testing (or use). The reports are stored in a database called defect database. It is the developers' job to take open, i.e. not yet fixed, defects from the database, analyze the corresponding reports, and fix the defects. Defect reports include many details about the corresponding defects including an *id* that uniquely identifies the defect, the *status* of the defect (e.g. new, verified, resolved), a *sum-*

mary field, and a *description* field. The description field is the richest source of information about the defect. The field describes in plain natural language details about the defect, including symptoms and steps to reproduce the defect. The summary field is an one-sentence description of the problem.

We propose here advanced methods for analyzing defect reports that take advantage of the description and summary fields of the reports. Our approach is to use advanced natural language processing (NLP) and informational retrieval (IR) techniques for automatic analysis of the reports. We regard each defect report as a textual document and use a well-known technique in IR, called the *vectorial representation*[1], to represent documents.

In our experiments, the K-means clustering algorithm proved to be by far the most successful (we tried several other clustering algorithms, e.g. EM, FarthestFirst available in Weka, a machine learning toolkit[15]) to find clusters of similar defects and thus the paper focuses on reporting the results obtained with this algorithm. We used two models to represent defect reports, one based on the summary field and another based on the description field of reports. Our experimental data consisted of defect reports collected from the open source Mozilla project (www.mozilla.org) but the proposed methods are transferable to defect reports from other projects, e.g. Eclipse (www.eclipse.org). The clustering was evaluated based on reports describing the same underlying problem. That is, defect reports are in the same cluster if they describe the same underlying problem. As a preview of our results, we found that using the summary field of defect reports for clustering (average accuracy=44.2240% when compared to human judgment) is better than using the much longer description field (average accuracy=29.3581%). We could say that given a set of defect reports we could identify the subsets, i.e. clusters, referring to the same underlying bugs with an accuracy of 44.2240%.

The rest of the paper is organized as follows. In the next section, *Related Work*, we outline previous efforts that are relevant to our work. The *Defect Management* section offers details about how defect reports are handled in the open source Mozilla project and its associated defect tracking tool Bugzilla. In the following section, *Experiments and Results*, we present details about our experiments and the results obtained. A *Discussion and Future Work* section follows. The *Conclusions* section ends the paper.

2. Related Work

There are two major lines of previous research relevant to our work. First, there is work on defect clustering. Second, there is research on using NLP and IR to mine artifacts from software repositories and in particular to analyze defect reports for various purposes.

Clustering is the unsupervised classification of data points (usually represented as vectors in a multidimensional space) into groups (clusters) based on similarity. A cluster is therefore a collection of objects which are *similar* to each other in the same cluster and are *dissimilar* to objects belonging to other clusters. The clustering problem has been addressed in many contexts and by researchers in many disciplines. This reflects the broad appeal of clustering and its usefulness as one of the steps in exploratory data analysis. While we are not aware of any particular work on clustering defect reports, there is published research related to clustering defects in the manufacturing of semiconductors[4] and integrated-circuits (IC; [14]). Karnowski and colleagues [4] showed that fuzzy logic can help better cluster defects on semiconductor wafer maps. Singh and Krishna[14] have shown that using clustering information in optimization testing can significantly improve the shipped product contrary to the previous assumption that the probability of a test to detect a faulty circuit is independent of the number of faults in that circuit.

The usage of natural language processing applications to improve software development and testing has been around at least since 1990s [13, 6, 7, 2].

More recently, there has been renewed interest in applying natural language techniques to mine useful artifacts from the various repositories associated with software projects (see the yearly Workshop on Mining Software Repositories at http://msr.uwaterloo.ca). We discuss next a series of research efforts that are directly related to our work on clustering defect reports.

Linstead and his colleagues [5] described a framework to automatically mine developer contributions and competencies from a given code base. They also used the framework for extracting software function in the form of topics. Their findings indicate that it is feasible to extract the function (in the form of topics) of source code and developer expertise on these topics. This information could be used, for instance, to better plan defect fixes: the most qualified developer will be assigned to handle defects related to topics s/he is expert in. Linstead and colleagues [5] treated source code as text.

The use of the vectorial representation[1] to address defect report related issues has been explored by Runeson, Alexandersson, and Nyholm [11] for the task of identifying duplicate defects (see also the work of Rus and Shiva [12]). It is noted that identifying duplicate defect reports is not exactly the same with clustering defect reports. Runeson and colleagues adapted ReqSimile (a tool that links customer wishes to product requirements using statistical natural language processing [10]) to identify duplicate defects. ReqSimile uses the vector space model and cosine similarity [1] to decide how related two requirements documents are.

The vector space model represents documents in a collection as vectors of V dimensions, where V is the vocabulary, i.e. the set of distinct words, of the collection. There is one dimension for each word in the vocabulary. Entries in documents' vectors are weights which indicate how important the corresponding word/dimension is for distinguishing the content of the document from other documents. If a word does not occur in a document, the corresponding weight is zero. The similarity of two documents is given by the cosine of the corresponding vectors, which can be seen as the normalized dot product of the vectors. Runeson, Alexandersson, and Nyholm [11] used the term frequency (TF) of words in a document as the weighting scheme. In this paper, we use a more powerful weighting scheme, namely Term Frequency-Inverted Document Frequency (TF-IDF; see the *Experiments and Results* section). Their experiments were conducted on Incident Reports, i.e. defect descriptions, from software projects at Sony Ericsson Mobile Communications. For each set of duplicate descriptions they identify a master defect. Their evaluation focuses on how well their method can retrieve the master defect in the list of top N most similar defects to a given defect description. Here, we use the vectorial representation with TF-IDF weighting for the task of clustering defect reports.

3. Defect Management

During testing, defects within software are discovered through testing (and fixed) and new functionality is added, which must be tested. The testers report defects using a defect management tool, also called defect tracking tool, whose back-end is typically a relational database. The general process of handling defect reports includes the following steps: the defect is found and filed in the defect tracking tool, the report is evaluated by an analyst, the defect is assigned to a developer, the developer finds the defect and fixes it, and the defect report is closed. The tester, analyst, and developer could be same or different persons depending on the size of the project. In open source projects, users voluntarily report defects.

3.1 Defect Handling in Mozilla

It is important to understand the details of defect handling in the Mozilla open source project because our experimental data is collected primarily from Mozilla's Bugzilla. Bugzilla, a bug tracking tool, allows testers to report bugs and assign these bugs to developers. Developers can use Bugzilla to keep a to-do list as well as to prioritize, schedule, and track dependencies. Not all entries in Bugzilla are bugs. Some entries are Requests For Enhancement (RFE). An RFE is a report whose severity field is set to enhancement.

Ideally, before reporting a defect, the tester must reproduce the bug using a recent build of the software, to see whether it has already been fixed, and search Bugzilla to check whether the bug has already been reported. If the bug can be reproduced and no one has previously reported it, the tester can file a new defect report including: the component in which the defect exists, the operating system on which the defect was found, a quick summary of about 60 or less characters, a detailed description of the defect, and attachments, for instance screenshots. We focus next on the summary and description fields (as presented in Mozilla's Bug writing guidelines), the two information-rich fields of any defect report.

A good summary should quickly and uniquely identify the defect. It should explain the problem, not the suggested solution. Example of a good summary is *Canceling a File Copy dialog crashes File Manager*, while bad examples are *Software crashes* and *Browser should work with my web site*.

The description field is a detailed account of the problem. The description field of a defect report should contain the following major sections, although the breakdown of the field into these sections is not enforced in Bugzilla: *overview* (more detailed restatement of summary), *steps to reproduce* (minimized, easy-to-follow steps that will trigger the bug; including any special setup steps), *actual results* (what the application did after performing the above steps), *expected results* (what the application should have done, were the bug not present), *build date* & *platform* (date and platform of the build in which you first encountered the bug), *additional builds and platforms* (whether or not the bug takes place on other platforms or browsers, if applicable), and *additional information* (any other useful information).

Any deviation from the above guidelines leads to vague reports which in turn lead to a less efficient process of handling the defects. On the other hand, recording every detail about a defect can lead to overkill. The reality is that seldom defect reports include all the above suggested details. In this paper, we present experiments on defect reports as collected from Mozilla's Bugzilla. In general, the collected reports are of good quality but reports of lower quality can be found among the collected reports.

4. Experiments and Results

We present in this section details on applying the K-means clustering algorithm to cluster defect reports. The results of the experiments on data from Mozilla's Bugzilla are discussed.

Clustering is the unsupervised classification of patterns (usually represented as a vector of measurements, or a point in a multidimensional space) into groups (clusters) based on

similarity. In other words, it is *the process of organizing objects into groups whose members are similar in some way*. A cluster is therefore a collection of objects which are *similar* to each other in the same cluster and are *dissimilar* to the objects belonging to other clusters.

Typical clustering involves the following steps [3]: data representation (optionally including feature extraction and/or selection), definition of a similarity measure appropriate to the data domain, clustering or grouping, and assessment of output.

We will address in more detail these steps next, starting with the *Data representation*, as a great deal of effort has been spent on this step.

4.1 How to Represent Defect Reports?

A first basic issue we must address is the logical view of the reports (see [1] more information on logical view). We may want to model the reports in the simplest way possible, for complexity reasons, while capturing the gist of the defect. The description field of a report is the most informative piece of information about the defect. On the other hand, it may contain too much information. Using the shorter summary of a report may lead to a more efficient algorithm for clustering as fewer words need to be considered to capture the meaning of the report. We experimented with both models, description vs. summary, in this paper and report the winner.

Another important issue is the formalism used for the representation. We used the vector space model [1]. The vectorial representation of documents was briefly described earlier in the *Related Work* section. A key feature in the vector space model is the weighting scheme of the words. We used the TF-IDF scheme, a composed measure that is the product of the frequency of a term in a document (TF), and its inverted document frequency (IDF)[1]. The basic idea of the TF-IDF measure is that a word is important, has a large weight, if it occurs frequently in the document (high TF) and if it does not occur in too many other documents (low document frequency which means high inverted document frequency[IDF]).

4.2 Preprocessing

Each defect report must be preprocessed before it is mapped onto the vectorial representation. Preprocessing maps a report onto a list of tokens that have linguistic meaning, i.e. words. It is comprised of the following steps: tokenization, stop word removal, and lemmatization. Tokenization, a well defined step in natural language processing, separates punctuation from words. Another important preprocessing step is to remove stop words. We used a standard list of stop words, e.g. the SMART list (ftp://ftp.cs.cornell.edu/pub/smart/english.stop). Lemmatization is another preprocessing step coming after stop word removal. It maps each morphological variation of a word in a text fragment to its base form, e.g. *go, going, went, gone* are all lemmatized to *go*, their root or base form. We used the WordNet lemmatizer available in the WordNet library[9].

[1]The IDF of a word is the percentage of distinct documents the word appears in from a very large collection of documents. It is used to measure the specificity of the word. The fewer documents a word occurs in, i.e. the rarer, the more specific the word is.

4.3 Clustering Experiments

Clustering software defect reports can take different forms. For instance, the clustering can be based on either the severity of the bug, or based on the fact that the defect reports describe the same defect, i.e. they are duplicates, or based on other criteria such as component/feature-specific clustering, e.g. clustering printer-related defect reports.

In this paper, we chose to defect reports based on their describing the same underlying bug. We regard a defect report and its duplicates as a cluster. This modeling is adequate as the original defect report should be similar, content-wise, to its duplicates. The data used in our experiments comes from Mozilla's Bugzilla where duplicate information is available. The duplicates are marked as such by members of the Mozilla development team and thus we deem them highly reliable. We automatically collected our experimental data from Mozilla's Bugzilla database as described next.

To create our data set, we started collecting 20 Bugs from the *Hot Bugs List* of Mozilla's Bugzilla. The *Hot Bugs List* contains the most filed recently bugs. The list can be sorted based on the number of duplicates (see the *Dupes* field for each entry in the *Hot Bugs List*). A secondary criterion for selecting the reports was the severity of the reports because we wanted to have diversity among the clusters in terms of severity. Thus, we finally chose the top 20 defect reports from the *Hot Bugs List* in terms of largest number of duplicates and diversity of severity. We retrieved 50 duplicates for each of the 20 defect reports. Hence, the total number of bugs considered were 1020. Only the top 20 bugs from the *Hot Bugs List* were chosen because only these bugs had more than 50 duplicate each. Having fewer than 50 duplicates for each original bug would have led to too small clusters. The list of 1020 bugs served as input to the Data Collection module of our system that automatically collects over the Internet (from Mozilla's Bugzilla database) the *Description* and *Summary* of these 1020 bugs and stores them locally in text files. The final data set contained 1003 data points because we eliminated 17 reports which had no proper description fields. For these eliminated reports the description field was empty or was simply redirecting the

user to another bug, e.g. the field contained textual pointers such as "*see bug #123*".

The data set was further processed and analyzed with two data mining tools: RapidMiner[8] and Weka[15]. We used RapidMiner to generate the vectorial representations for defect reports. Weka was used to apply the K-means clustering algorithm to the data set. The two processing steps are explained in more details below.

The **TF-IDF Generation Module** computes the TF-IDF weights for each term in the vocabulary. For *Description* documents, i.e. reports represented using their *Description* field, the vocabulary size was 4569. The vocabulary size indicates the number of dimensions of the document vectors in the vectorial representation. For *Summary* documents, the vocabulary size was 991. From the vocabulary size of the two representations, description vs. summary, we notice the efficiency of the summary-based representation which has lower dimensionality.

In the next step, the defect reports in vectorial representation must be mapped to a format that WEKA requires in order to do clustering. The required format is the ARFF (Attribute Relation File Format) file format. This file can be generated using the RapidMiner's *ArffExampleSetWriter* operator.

Clustering. There are two major categories of clustering algorithms as listed below.

- *Hierarchical clustering* algorithms produce a nested series of partitions based on a criterion for merging or splitting clusters based on similarity.
- *Partition based clustering* algorithms identify the partition that optimizes (usually locally) a clustering criterion.

Example algorithms from each category are hierarchical agglomerative (HAC) and K-means, respectively [15]. HAC produces a hierarchical structure of clusters while K-means leads to a flat, direct clustering. In HAC, each data point is initially regarded as an individual cluster and then the task is to iteratively combine two smaller clusters into a larger one based on the distance between their data points. In the K-means algorithm, we specify a priori the number of clusters (K) we would like to have in the end. In testing, this could be useful when testers want to find out what are K, say K=20, clusters in the set of open defects. The algorithm usually starts with K seed data points which are considered as individual clusters. In subsequent iterations, the remaining data points are added to some cluster based on the distance to the centroid of each cluster. The centroid is an abstract data point of an existing cluster that is found by averaging over all the other points in the cluster. A distance metric must be defined for clustering algorithms. The advantage of using K-means is that it is very easy to implement and relatively efficient i.e. $O(t \times k \times n)$, where n is the number of objects, k is the number of clusters and t is the number of iterations. In many cases, the $k, t \ll n$ and hence can be ignored.

K-Means Clustering with Weka. We used the Simple K-Means algorithm in WEKA to obtain the clusters and automatically evaluate the performance of the clustering. Some implementations of K-means only allow numerical values for attributes. In case of categorical attributes they must be converted to numerical values. It may also be necessary to normalize the values of attributes that are measured on substantially different scales (e.g., *age* and *income*). WEKA's SimpleKMeans algorithm automatically handles a mixture of categorical and numerical attributes. Furthermore, the algorithm automatically normalizes numerical attributes when doing distance computations. The WEKA's SimpleKMeans algorithm uses Euclidean distance measure to compute distances between instances and clusters. To perform clustering, we needed to set a couple parameters: *number of clusters*, which informs the clustering algorithm how many clusters to generate, and *seed*. The seed value is used in generating a random number which is, in turn, used for making the initial assignment of instances to clusters. In general, K-means is quite sensitive to how clusters are initially assigned and thus it is often necessary to try different values and evaluate the results.

4.4 Results

The clustering *accuracy* is the number of documents correctly clustered divided by the total number of documents. WEKA includes an evaluation module that automatically compares the output of the clustering algorithm to the correct clustering, which in our case are the expert human judgments regarding whether one description is a duplicate of another as indicated in Mozilla's Bugzilla.

The performance of the K-means clustering algorithm depends on the number of seeds initially used to start the clustering. We experimented with various values for the *seed* parameter to find what is the best number of seeds to use. We varied the seed value from 0 to 1003 in increments of 1. For *Description*-based representation of reports, the maximum performance was found to be for the seed value 33, which is 47.7567%, and the minimum performance was found for the seed value 175, which is 7.2782%. The average performance obtained was about 29.3581%.

For *Summary*-based representations, the maximum performance was found for seed value 825, which is 59.8205%, and the minimum performance was for seed value 275, which is 34.2971% . The average performance obtained was about 44.2240%. Thus, our proposed method is able to identify clusters of reports describing the same underlying problem with an accuracy of 44.2240%. A baseline approach would be to always guess one of the twenty clusters

for an average accuracy of 20%. A statistical t-test (α=0.05) was used to compare the baseline to our approach and found our approach to be significantly (p¡0.0001) better than this baseline. The null hypothesis was that the average accuracy computed over 1004 seed points is equal to the average accuracy of the baseline approach.

5. Discussion and Future Work

Having a clustering feature integrated in a defect tracking tool such as Bugzilla could be extremely beneficial. For instance, in a large software development project this feature can be used periodically, e.g. once a week or once a month, to analyze the set of open defects by clustering them. The clusters can be used by the testing team in various ways, for instance to prioritize their work. We plan to continue our investigation of clustering defect reports by using other representations for defect reports, e.g. using only the *overview* section of the description field of a software report. One interesting research question to be explored in the future is the suitability of our proposed methods to cluster similar but not necessarily identical defect reports.

6. Conclusions

We addressed in this paper the challenging task of clustering defect reports. The evaluation was based on clusters containing defect reports describing the same underlying bug. Our experiments on defect reports from Mozilla's Bugzilla with the K-means clustering algorithms showed that using reports' summaries together with a TF-IDF vectorial representation leads to better clustering than using full descriptions of reports, which is also computationally more expensive.

7 Acknowledgments

This research has been supported by a grant from the Systems Testing Excellence Program (STEP) of The University of Memphis. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of The University of Memphis.

We are also extremely grateful to Sameer Mohammed for his help with implementing this project as part of his Masters Project.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[2] L. Etzkorn, L. Bowen, and C. Davis. An approach to program understanding by natural language understanding. *Natural Language Engineering*, 5(1):1–18, 1999.

[3] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[4] T. P. Karnowski, S. S. Gleason, and J. Kenneth W. Tobin. Fuzzy logic connectivity in semiconductor defect clustering. Technical report, Oak Ridge National Laboratory.

[5] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining eclipse developer contributions via author-topic models. In *International Workshop on Mining Software Repositories*, Minneapolis, USA, May 19-20 2007.

[6] P. Lutsky. Documentation parser to extract software test conditions. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 1992.

[7] P. Lutsky. Using a document parser to automate software testing. In *Proceedings of the 1994 ACM Symposium on Applied Computing*, pages 59–63, Phoenix, Arizona, 1994.

[8] I. Mierswa, M. Wurst, R. Klinkenbergand, M. Scholz, and T. Euler. Yale: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06)*, 2006.

[9] G. Miller. WordNet: a lexical database for english. *Communications of the ACM*, pages 39–41, 1995.

[10] J. N. och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell. Speeding up requirements management in a product software company: Linking customer wishes to product requirements through linguistic engineering. In *International Conference of Requirements Engineering*, 2004.

[11] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th International Conference on Software Engineering*, 2007.

[12] V. Rus and S. Shiva. A general framework for quantitative software testing. In *Proceedings for the First International Workshop on Advances and Innovations in Software Testing*, Memphis, USA, May 6-8 2007.

[13] J. Schlimmer. Learning meta knowledge for database checking. In *Proceedings of the National Conference of the American Association of Artificial Intelligence (AAAI'91)*, pages 335–340, 1991.

[14] A. Singh and C. Krishna. On the effect of defect clustering on test transparency and ic test optimization. *IEEE Transactions on Computers*, 45(6):753–757, June 1996.

[15] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

Subjective Assessment of the Mutual Influence of ISO 9126 Software Qualities: An Empirical Study

Sandro Morasca
Dipartimento di Scienze della Cultura, Politiche e dell'Informazione
Università degli Studi dell'Insubria
Como, Italy
sandro.morasca@uninsubria.it

Abstract— **It is usually believed that software qualities may influence each other and that improving one software quality may cause some other qualities to worsen. Since no commonly used set of metrics exist for quantifying the software qualities of interest, the perception of the influence of one software quality on another is often used in practice when making decisions about software development. We describe the results of an empirical study on the beliefs of software practitioners in the mutual influence of the ISO 9126 software qualities. The study shows that the subjects believed, on average, that a number of influences exist between the ISO 9126 software qualities. However, the practitioners did not believe that trade-offs exist among several of these software qualities. In other words, they believed that improving one quality is often likely to improve other qualities as well.**

Keywords-ISO 9126;software quality

I. INTRODUCTION

During software development, decisions are made to obtain better software products in terms of several software qualities. Software qualities (or attributes) are usually divided into internal and external software ones:

- Internal software qualities (e.g., size, structural complexity, coupling) can be measured based on the knowledge of the software product alone;
- External software qualities (e.g., reliability, usability, maintainability, and readability) are related to both the software product and its environment. External software qualities are related to the many "users" of a software product, e.g., reliability and usability are related to the software's final user; maintainability and readability are related to the software developers.

External software qualities are usually the qualities of industrial interest, since they are related to the behavior of a software product towards its "users." On the other hand, the study, measurement, and "improvement" of an internal software quality is fully justified only if that internal software quality is believed to be linked to least one external software quality of industrial interest or to a process quality (e.g., cost, time-to-market). For instance, one of the goals of the introduction of object-oriented software development techniques was to obtain software systems divided into modules that were "better" in terms of their degree of internal cohesion and external coupling, i.e., with a higher degree of internal cohesion and a lower coupling with each other. However, higher cohesion and lower coupling were not the final objective of the introduction of object-oriented techniques: higher cohesion and lower coupling were supposed to improve the external qualities of software systems, e.g., in terms of maintainability, reusability, error-proneness, etc. Thus, even when they are based on some internal software quality, decisions on software development techniques and processes are actually often made to improve of some external software quality. Furthermore, even when the goal is to improve some process quality, decisions do affect external software qualities.

In this paper, we focus only on external software qualities, so the term "software quality" is used in this paper with the meaning "external software quality," unless otherwise explicitly stated.

It is commonly believed that there may be interactions among software qualities, and that improving one of the qualities of a software product may have a (possibly adverse) effect on other software qualities. It is argued that, if this was not the case, then it would be possible to improve all software qualities at the same time. This entails that practitioners must constantly make decisions as to how to proceed during all software development phases based on a balancing among the different software qualities that are affected by the decisions. For instance, a design decision may increase the level of one quality while decreasing the level of another. Therefore, the interactions among software qualities should always be taken into account during the software development process.

In principle, decision making should be based on quantitative information, if possible. This requires:

- the definition of adequate measures for the qualities of interest, and
- empirical studies that ascertain or at least provide convincing support on the existence and nature of the interactions among software qualities.

However, the state of the art of software measurement has not reached a point in which (1) adequate measures are often available for the qualities of interest, and (2) a sufficient number of studies are available on the interactions of software

qualities. As a consequence, software designers and developers often make decisions based on their own beliefs in the existence and nature of interactions among software qualities.

The goal of the study documented in this paper is to assess the degree of belief of software practitioners on the interactions among the ISO 9126 software qualities, and determine if there is some degree of consensus among the practitioners. This paper takes an empirical approach, in that its results are based on data collected from practitioners in a series of empirical studies carried out with practitioners from Italian and Swiss software companies. We wanted to obtain a characterization of software practitioners' beliefs about the mutual influences of software qualities, i.e., have a quantitative idea of the distributions of these beliefs and descriptive statistics such as mean, standard deviation, and median. In addition, we also explored the significance of these beliefs are from a statistical point of view. At any rate, this study provides initial evidence, but more studies are certainly needed to gather more evidence.

The remainder of the paper is organized as follows. Section 2 concisely describes the ISO 9126 standard. Section 3 describes the setting of the empirical study. The statistical hypotheses we tested are in Section 4. The empirical results are reported and discussed in Section 5. Section 6 concisely reports on the internal, external, and construct validity of the empirical study. Conclusions and outline of future work are in Section 7.

II. ISO 9126

A few quality models have been defined in the last few years, starting from [9]. Quality models are often defined in a hierarchical fashion (e.g., see Fig. 1). The overall quality of a software product is viewed as composed of a set of qualities. These qualities may be hierarchically refined by several layers of qualities (some of which may be internal software qualities). At each layer, the relative importance of each quality may be weighted depending on the measurement problem at hand. This hierarchical refinement process ends with the definition of a set of measures to quantify each quality. The weights provided to each quality may be used to combine the values of the measures into a single value for the overall software quality.

In some quality models, the qualities and measures associated with all the nodes of this hierarchical structure are fixed [9]. In other quality models, some parts are left tailorable/unspecified, for instance the set of measures [5]. There are also general, flexible framework that can be tailored to several different environments and measurement objectives, like the Goal/Question/Metric paradigm [1,2], which allows for the building of different quality models and measures. Depending on the measurement application at hand, different qualities may be chosen to be studied. Also, the same quality may be refined via different sets of qualities in different measurement applications. At the end of the hierarchical generation process, different measures may thus be generated for different measurement applications.

At any rate, like in many other engineering disciplines, efforts have been made in recent years to reach a standard view of software quality and of a quality model. This has led to the definition of the ISO 9126 quality model [5, 6, 7], in which software quality is viewed as composed of 6 qualities.

The ISO 9126 [5] standard was originally defined in 1991. Refinements and guidelines have been provided over the years. For instance, standard sets of external and internal measures have been provided in the new version [6, 7] of the ISO 9126 standard and a measurement process-oriented view has been introduced in the ISO 14528 standard. However, the basic structure of the ISO 9126 standard has not been substantially affected by the its evolution. Thus, we have used the original standard in this study.

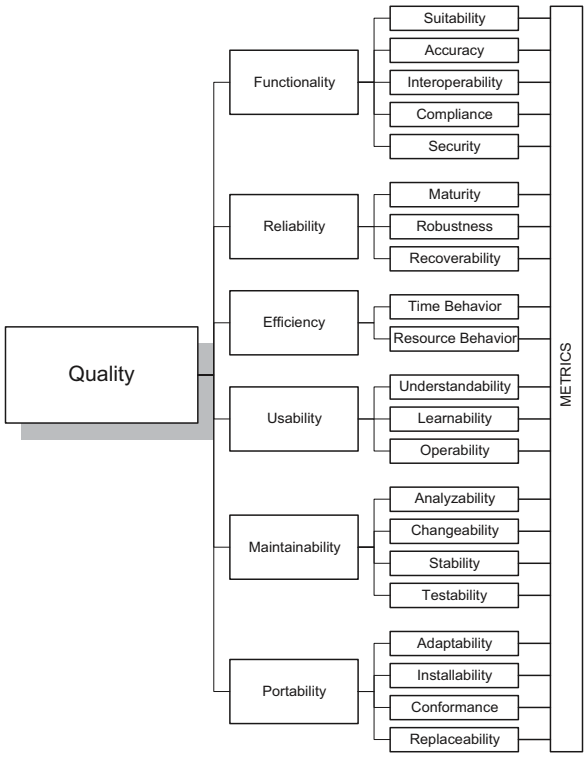


Figure 1. The layers of ISO9126.

The ISO 9126 standard is organized on four layers (see Fig. 1). The first layer is that of the overall software quality. On the second layer, the overall software quality is refined as being composed of six qualities. Each quality is refined in terms of a set of qualities on the third layer. The qualities defined on the third layer afford direct quantification via a set of measures, which are left unspecified in the original standard. For completeness, we now list the six qualities on the second layer along with their refining qualities on the third layer.

- Functionality is about meeting stated and implied needs when the software is used under specified conditions.
- Reliability is about maintaining the level of performance under specified conditions.
- Efficiency is about providing the required performance relative to the amount of resources used, under stated conditions.

- Usability is related to being able to understand, learn, and use the software under specified conditions.
- Maintainability is related to analyzing software products to find a fault, change the software products, making sure the change does not have side-effects, and testing the new version.
- Portability is related to being able to make a software product run in different environments.

The ISO 9126 standard also says that these qualities describe, with minimal overlap, software quality. Thus, they are somewhat orthogonal, even though this does not mean that there are no dependencies among them. However, some degree of independence should be there. If there was a very strong influence between two such qualities, one may wonder why they are both in the standard. This is what happens in other fields as well. When a strong statistical correlation is found between two variables, then the two variables basically contain the same information, since, once one of them is known, the other one is basically known too, i.e., it can be discarded from the core modeling of some phenomenon as being redundant.

ISO 9126 is therefore a general reference model that lists a number of qualities that software practitioners should have in mind when they build software. Practitioners should be aware that interactions among these qualities may exist, so they can make informed decisions during software development. Here, we focus on how aware practitioners are of these interactions.

III. Setting of the Empirical Study

The main goal of the empirical study was to evaluate the degree of consensus about the influence of each second-layer ISO 9126 quality on the others. The empirical study involved 145 software practitioners from Italian and Swiss software development companies, who were attending advanced classes on software quality. The subjects were involved in a variety of different application domains, from business software applications, to web applications, to automotive, etc. The data were collected during ten editions of the same course on software quality, over a period of 4 years, from 2000 to 2004.

The ISO 9126 standard was first explained to the subjects. Then, the subjects were asked to quantify their personal degree of belief about how strongly each ISO 9126 quality influences the other ISO 9126 qualities and data were collected. Data were collected by means of a very simple data collection form, organized in the shape of a matrix. Each cell of the matrix was related to a pair of qualities, one of which was the "influencing" quality, say *X*, while the other was the "influenced" quality, say *Y*. The idea was to check whether it was believed that pushing for quality *X* (the "influencing" quality) would also affect quality *Y* (the "influenced" quality), as both qualities depend on the same root cause, i.e., the way the software is written. We distinguished an influencing from an influenced quality because the influence of *X* on *Y* may not necessarily be of the same type (i.e., positive or negative) and strength as the influence of *Y* on *X*.

For each pair of qualities <*X*, *Y*>, the subjects were asked to provide a value on a –5 .. +5 scale, where the value –5 has the meaning "*X* strongly negatively influences *Y*" and the value +5 has the meaning "*X* strongly positively influences *Y*." Thus, value 0 has the meaning "*X* does not influence *Y*." The -5 .. +5 scale was chosen because (1) it would clearly represent both negative and positive influences and (2) a sufficient number of values were available to respondents to quantify their beliefs about the degrees of influence and provide different rankings for different degrees. It was made clear to the subjects that the single values of the scale were not important *per se*, but they were used to obtain a ranking of the beliefs of the various influences. For instance, the value '4' *per se* did not provide much information, but it indicated that the subject believed a higher degree of influence than a '2' value. The subjects were also allowed to leave blank the values associated with a pair <*X*, *Y*>, if they did not feel sufficiently confident about providing any value. This was done to reduce the number of unsubstantiated guesses. For completeness, it was explained to the subjects that "blank" and 0 had two different meaning, i.e., "blank" meant "don't know," while 0 actually meant "I believe there is no influence." It was made clear to the subjects that there was no "correct answer" for any of the pairs <*X*, *Y*>. The aim of the survey was only to capture the participants' beliefs in a quantitative form and no consequences of any kind would derive to them from their answers. In addition, the survey also had the pedagogical goal [3] of making the practitioners reflect about the ISO 9126 standard, the meaning of its qualities, their interactions, and the fact that software quality needs to be planned in advance and that development decisions may have an impact on several different qualities.

IV. Statistical Hypotheses Explored

The hypotheses we tested are of two kinds, as follows.

- Hypotheses related to single distributions. Since the data we collected were ordinal [10], we investigated hypotheses related to the median of each distribution, instead of the mean, which, strictly speaking, is not a meaningful indicator of central tendency for ordinal data [10]. The motivations for these hypotheses related to single distributions come from conventional wisdom that has it that qualities do influence one another. We wanted to test whether our respondents actually believed in this idea. Given two qualities *X* and *Y*, our null hypothesis was H_0: $M_X = M_Y$, and the alternative hypothesis was H_1: $M_X \neq M_Y$, where M_X and M_Y represent the medians of *X* and *Y*, respectively. As explained in Section III, these hypotheses have the meaning "if software is written in such a way as to push for quality *X*, I believe that quality *Y* is also affected." Since no assumptions on the direction (positive or negative) of influence could be safely made, our above statistical hypotheses were nondirectional, and we used two-tailed statistical tests to test them [8]. Note that two-tailed statistical tests are more conservative than one-tailed tests for a given level of statistical significance, so it is less likely to reject the null hypothesis with two-tailed tests. Thus, we wanted to be even more careful about investigating the existence of such beliefs.

- Hypotheses related to the associations of pairs of distributions. Given two qualities *X* and *Y*, we checked whether there was an association between the ranking provided by the subjects about the influence of *X* on *Y* and the influence of *Y* on *X*, i.e., the influence between *X* and *Y* works in both directions. For each association checked, the null hypothesis states that there is no association, while the alternative hypothesis states that there is a positive association, since one may expect that if it is believed that there is a positive (resp. negative) influence of *X* on *Y*, then there is a positive (resp. negative) influence of *Y* on *X*. Studying these associations is an additional way of checking whether the subjects believed that (1) there is a real trade-off between qualities (this is the case when a double negative association exists between the distributions, i.e., improving *X* makes *Y* worse, and improving *Y* makes *X* worse), or (2) the two qualities *X* and *Y* support each other (this is the case of a double positive association between the distributions, i.e., improving *X* also improves *Y*, and *vice versa*).

V. RESULTS

Here, we first provide the results on the hypotheses related to single distributions are in Section V.A. The results related to associations between distributions are in Section V.B.

A. Results Related to Single Distributions

Table I contains the summary statistics for the data collected for each of the possible pairwise interactions between ISO 9126 qualities. Each ISO 9126 quality is identified by its initial letter. The rows represent the influencing qualities, while the columns represent the influenced qualities. Each cell of the matrix in Table I contains the values of:

- *N*: the number of values collected
- *1Q*: the location of the first quartile
- *M*: the median of the data distribution (the value 0.5 in cell <*F*,*U*> is due to the midpoint approximation used when a distribution has two medians)
- *3Q*: the location of the third quartile
- *m*: the average value of the data distribution
- σ: the standard deviation of the data distribution
- *p*: the statistical significance of the Wilcoxon signed rank test [11], used to test the hypothesis about the believed influence between the two qualities. The Wilcoxon signed rank test is appropriate for assessing hypotheses about the medians of single data distributions of discrete, ordinal data.

We have also reported the values of *m* and σ to provide a more complete idea of the central tendency and spread of the data distributions, though the data were collected on an ordinal scale, for which *m* and σ are not appropriate, in principle.

As a first observation, Table I shows that the 145 subjects provided values for most of the cells. The cell with the highest number of values (135) corresponds to pair <*F*,*M*> and the cell with the lowest number of values (117) to pair <*U*,*R*>. This seems to indicate that most subjects had a sufficiently solid idea about most interactions among software qualities.

Second, the data distributions show a fairly large spread of values. As could be expected with 145 total respondents, all of the 30 data distributions actually have the maximum spread possible, i.e., -5 .. +5 (this piece of information is not reported in Table I, because it would be the same for all cells). As for the interquartile range, the average of the distance between *1Q* and *3Q* is 4.43 and the standard deviation is 1.07. In addition, all of the standard deviations of the distributions are between 2.31 and 3.06. These measures of spread show that the participants' consensus about the influences between ISO 9126 software qualities was limited and some of the participants believed that there was a really strong interaction between software qualities, but their beliefs were conflicting.

TABLE I. STATISTICS FOR THE DISTRIBUTIONS

	F	R	E	U	M	P
F		N: 133 1Q: -2 M: 0 3Q: 3 m: 0.82 σ: 2.73 p:<0.001	N: 131 1Q: -2 M: 0 3Q: 3 m: 0.53 σ: 2.82 p:<0.001	N: 130 1Q: -1 M: 0.5 3Q: 3 m: 0.65 σ: 2.88 p:<0.001	N: 135 1Q: -1 M: 2 3Q: 3 m: 0.99 σ: 3.02 p:<0.001	N: 125 1Q: -2 M: 0 3Q: 1 m:-0.38 σ: 2.45 p:<0.001
R	N: 126 1Q: 0 M: 0 3Q: 3 m: 0.81 σ: 2.49 p:<0.001		N: 136 1Q: -1 M: 2 3Q: 4 m: 1.32 σ: 2.89 p:<0.001	N: 129 1Q: -1 M: 0 3Q: 3 m: 0.46 σ: 2.71 p:<0.001	N: 127 1Q: 0 M: 2 3Q: 3 m: 1.50 σ: 2.51 p:<0.001	N: 118 1Q: -1 M: 0 3Q: 0 m:-0.22 σ: 2.31 p:0.052
E	N: 123 1Q: -1 M: 0 3Q: 2 m: 0.29 σ: 2.35 p: 0.45	N: 128 1Q: 0 M: 1 3Q: 3 m: 1.04 σ: 2.55 p:<0.001		N: 127 1Q: -2 M: 0 3Q: 2 m: 0.05 σ: 2.83 p:<0.069	N: 126 1Q: -2 M: 0 3Q: 2 m: 0.01 σ: 2.81 p:<.145	N: 125 1Q: -3 M: -1 3Q: 0 m:-1.16 σ: 2.52 p:<0.001
U	N: 122 1Q: 0 M: 0 3Q: 3 m: 0.79 σ: 2.68 p:0.011	N: 117 1Q: 0 M: 0 3Q: 2 m: 0.39 σ: 2.36 p:0.076	N: 120 1Q: -2 M: 0 3Q: 2 m: 0.11 σ: 2.76 p:0.359		N: 129 1Q: -2 M: 0 3Q: 2 m: 0.09 σ: 2.73 p:0.409	N: 127 1Q: -3 M: 0 3Q: 1 m:-0.41 σ: 2.76 p:0.107
M	N: 122 1Q: -1 M: 0 3Q: 3 m: 0.43 σ: 2.68 p:0.148	N: 124 1Q: -1 M: 1 3Q: 3 m: 0.90 σ: 2.96 p: 0.003	N: 127 1Q: -2 M: 0 3Q: 2 m: -0.24 σ: 2.91 p:0.261	N: 125 1Q: -1 M: 0 3Q: 1 m: -0.03 σ: 2.40 p:0.213		N: 125 1Q: 0 M: 1 3Q: 3 m: 1.03 σ: 2.86 p:0.002
P	N: 118 1Q: -2 M: 0 3Q: 0 m:-0.67 σ: 2.40 p:0.056	N: 119 1Q: -2 M: 0 3Q: 1 m: -0.45 σ: 2.56 p:0.365	N: 125 1Q: -3 M: -2 3Q: 0 m: -1.22 σ: 2.55 p:<0.001	N: 127 1Q: -2 M: 0 3Q: 1 m: -0.23 σ: 2.83 p:<0.34	N: 126 1Q: -2 M: 0 3Q: 3 m: 0.32 σ: 3.06 p:0.425	

Third, notwithstanding the spreads of values, a number of distributions show that it makes sense to hypothesize the existence of some degree of consensus about the influence of some software qualities on other software qualities. To this end, the statistical significance of the results is quite clear. We set a statistical threshold of 0.05, as is customary in empirical software engineering. Based on the number of responses we

had for each distribution we could safely use the normal approximation to the exact distribution of *W*, the statistic used in the Wilcoxon signed rank test. Since the test is two-tailed, p-values less than 0.025 provided us with evidence to reject the null hypothesis. We have obtained statistically significant results in 15 distributions out of 30, i.e., in half of the cases, our respondents believed that two qualities interact.

Fourth, it is somewhat surprising that the majority of influence relationships are believed to be nonnegative. This can be assessed in several different ways. Out of 30 medians, 2 turned out to be negative, 21 null, and 7 positive. The interquartile range turns out to be more biased towards negative values (e.g., -2..1, like in the case of <*F,P*>) in 8 cases, centered on 0 (e.g., -2..-2 like in the case of <*U,E*>) in 6 cases, and more biased towards positive values (e.g., -1..3, like in the case of <*F,M*>) in 16 cases. Out of 30 relationships, 20 have a value $m > 0$ and 10 have a value $m < 0$. So, all of these statistics provide the same kind of evidence. These results are somewhat unexpected, since it is commonly believed that there is a trade-off among software qualities, i.e., trying to improve one of them may result in worsening another.

In addition, 8 out of the 10 distributions with a negative value for *m* are actually related to Portability, so only 2 out of the 20 distributions that do not involve Portability actually have a negative *m*. Based on these results, one may conclude that the respondents in the sample viewed Portability as a sensitive quality that needs to be taken into account during development, since decisions about software products to improve other qualities may negatively affect Portability.

B. Results Related to Associations between Distributions

Table II contains the values of Spearman's *rho* and Kendall's tau_b [4] about the agreement of the rankings of the influences <*X,Y*> and <*Y,X*>. We used Spearman's *rho* and Kendall's tau_b because the variables we used are ordinal ones. Both statistics range between –1 (perfect negative association) and +1 (perfect positive association), so they quantify the degree of association. In addition, both statistics can be used to check whether the association is statistically significant. We found that there is a statistically significant association (at the 0.05 level) between the rankings of the influence of quality *X* on quality *Y* and the rankings of the influence of quality *Y* on quality *X* for all pairs of qualities <*X,Y*>.

For a pair of qualities <*X,Y*> where both the influence of *X* on *Y* and of *Y* on *X* are statistically significant (see Table I) and the influence is of the same direction, these results show that the respondents seem to believe, even though with various degrees of strength, that there is a bidirectional influence between qualities. Specifically, the association with the highest values for *rho* and tau_b involves Usability and Portability. The subjects seem to indicate that there is a real trade-off between these two qualities, since the values of m for both <*U,P*> and <*P,U*> in Table I are negative and also the interquartile ranges are biased towards negative values. The second highest values for *rho* and tau_b involve Functionality and Reliability. This result could actually be somewhat expected, as this indicates that the subjects believed that the two qualities support each other, i.e., improving one will also improve the other.

Let us now look at pairs of qualities for which there is no conclusive evidence on the fact that one influences the other, but the influences in Table I are of the same sign, e.g., Efficiency and Functionality, though there is evidence that Functionality influences Efficiency, as shown by the p-values in Table I. Table II shows that there is an association between the scores of <*F,E*> and <*E,F*> which seems to confirm that, even though the influence <*E,F*> is not statistically significant, the two qualities in general are believed to somewhat support each other, or at least they are not conflicting anyway, even though one influence is stronger than the other. Further investigations may be needed in this case, though.

TABLE II. SUMMARY STATISTICS BY INFLUENCING AND INFLUENCED QUALITY

<X,Y>	<Y,X>	rho	tau_b
<F,R>	<R,F>	0.68	0.60
<F,E>	<E,F>	0.59	0.46
<F,U>	<U,F>	0.60	0.50
<F,M>	<M,F>	0.35	0.31
<F,P>	<P,F>	0.58	0.52
<R,E>	<E,R>	0.61	0.49
<R,U>	<U,R>	0.54	0.42
<R,M>	<M,R>	0.31	0.26
<R,P>	<P,R>	0.35	0.29
<E,U>	<U,E>	0.55	0.45
<E,M>	<M,E>	0.51	0.44
<E,P>	<P,E>	0.65	0.56
<U,M>	<M,U>	0.50	0.43
<U,P>	<P,U>	0.71	0.61
<M,P>	<P,M>	0.61	0.54

As for those pairs of qualities for which there is no evidence that either quality influences the other (e.g., see the cells for <*E,U*> and <*U,E*> in Table I), the results on *rho* and tau_b provide further support for the lack of actual influence.

VI. VALIDITY OF THE EMPIRICAL STUDY

Like in any empirical study, we need to examine the factors that may have biased our results. We believe that the following factors may have influenced an empirical study like ours.

A. Internal Validity

These two factors could pose a threat to the internal validity of the empirical study.

- *Subjects*. The subjects were not selected beforehand, so, no self-selection occurred, and various different professional figures were involved. This may be acceptable for the internal validity of our study.
- *Knowledge on ISO 9126*. All the subjects were given the same information about ISO 9126. It would not make sense to "randomize" this factor, i.e., provide the subjects with different degrees of knowledge about ISO 9126.

B. External Validity

The question may arise as to how representative our empirical study is in the population of empirical studies on ISO 9126 qualities.

- *Subjects*. It would not be possible to claim that the subjects were representative of the population of software project leaders and developers. However, no pre-selection was carried out and the class was given 10 times to different people with different backgrounds, skills, and expertise in different application domains.
- *Knowledge on ISO 9126*. Not all project leaders and developers are in general knowledgeable on ISO 9126, and even those who are may have various degrees of knowledge. However, some knowledge on ISO 9126 was an obvious precondition of the study, if only to have a common terminology across all subjects.

C. Construct Validity

We did not use any particular measure defined for the basic constructs, i.e., the ISO 9126 qualities, of the study. There is no general agreement on how these constructs should be measured, and any measures could be questioned as to whether they actually quantify the quality they purport to measure. An operational measure was defined for the assessment of the mutual influences among qualities. One may wonder whether this was an appropriate way to capture these influences. We chose an ordinal measure because it would have made little sense to choose an interval or ratio measure, since we are interested here in rankings. Also, we provided a measure with 11 integer values (from -5 to +5), so it was sufficiently fine-grained to capture the rankings among influences.

VII. Conclusions and Future Work

The research documented in this paper has investigated empirically whether several software project leaders and developers believed that external software qualities influence each other. The results seem to indicate that, on average, the subjects believed in the existence of a number of such influences, and, more surprisingly, on the fact that these influences are positive. Thus, the need for trade-offs between possibly conflicting qualities does not seem to be perceived by our respondents. Also, the influences seem to be bidirectional, which provides further evidence to this perceived lack of trade-offs. The one real exception is Portability, which is believed to be conflicting with almost all other qualities.

On the other hand, several reasons may provide an explanation for why no mutual influence was detected for a number of pairs of qualities, including:

- there is actually no influence in general for those pairs, i.e., the qualities are mostly independent of each other;
- the course on software quality did not explain adequately the concepts behind the ISO 9126 qualities;
- the ISO 9126 qualities should be provided with more precise explanations; as for this, the introduction of standard sets of measures will provide further aid in the understanding of the definition of the ISO 9126 qualities.

At any rate, a good deal of further work is will need to be carried out, including:

- gathering more data;
- understanding the reasons behind software practitioners' beliefs, so these reasons can be studied and tested;
- studying the effect of providing the new subjects with the new standard sets of measures for the qualities;
- investigating the relationships between internal software qualities and external ones.

Acknowledgment

The research presented in this article was partially funded by the IST project QualiPSo, sponsored by the EU in the 6th FP (IST-034763); the FIRB project ARTDECO, sponsored by the Italian Ministry of Education and University; and the project La qualità nello sviluppo software, sponsored by the Unversità degli Studi dell'Insubria.

References

[1] V. R. Basili and D. H. Rombach, "The Tame Project: Towards Improvement-Oriented Software Environments," IEEE Trans. Software Eng., vol. 14, no. 6, pp. 758-773, June 1988.

[2] V. R. Basili and D. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Trans. Software Eng., vol. 10, no. 11, pp. 758-773, November 1984.

[3] J. Carver, M.L. Jaccheri, S. Morasca, F. Shull, „Issues in using students in empirical studies in software engineering education," in IEEE METRICS, pp. 239–249, Sydney, Australia, 2003.

[4] J. D. Gibbons, Nonparametric Measures of Association, Sage Publications, 1993.

[5] ISO 9126, "Information technology - Software product evaluation - Quality characteristics and guidelines for their use," 1991.

[6] ISO/IEC 9126-1:2001- Software Engineering - Product Quality Part 1: Quality Model, ISO/IEC, 2001.

[7] ISO/IEC 9126-2:2002- Software Engineering - Product Quality Part 1: External Metrics, ISO/IEC, 2002.

[8] G. K. Kanji, 100 Statistical Tests, SAGE Publications, London, UK, 1999.

[9] J. A. McCall, P. K. Richards, G. F. Walters, Concepts and Definitions of Software Quality Factors in Software Quality, Vol. 1, Springfield, VA, USA: NTIS, November 1977.

[10] F. S. Roberts, Measurement Theory, Addison-Wesley, Reading, 1979.

[11] D.J. Sheskin, Handbook of Parametric and Nonparametric Statistical Procedures, 3rd ed. Chapman & Hall CRC, 2003.

Reverse Engineering Interface Protocols for Comprehension of Large C++ Libraries during Code Evolution Tasks

Edward B. Duffy, Jason O. Hallstrom and Brian A. Malloy
Computer Science Department
Clemson University
Clemson, SC 29634, USA

E-mail: {eduffy,jasonoh,malloy}@cs.clemson.edu

Abstract

In this paper, we describe our trace monitoring system and a methodology for reverse engineering interface protocols to capture the sequence of method invocations for large C++ applications. To evaluate our system, we present a case study using the networking library in the Mozilla Internet Application Suite, and three Mozilla applications: Firefox, Thunderbird and Sunbird. We use trace monitoring of the library to capture the interface protocols for the classes in the library and our preliminary results support our assumption that interface protocols follow a specific pattern and that these patterns can facilitate comprehension of the underlying interactions among the classes in the system.

1. Introduction

The process of software maintenance, including modification, refactoring, and usage of complex object oriented systems, requires knowledge about the system under study and, in particular, about the interactions among the classes and components of the system. However, software artifacts that describe these interactions are frequently unavailable and for large, open-source C++ applications, they are virtually nonexistent. Thus, much of the research in software engineering has focused on the development of tools to automatically generate information to improve comprehension of the application under study, and thereby facilitate the maintenance effort.

In this paper, we present Hylian, our system for code regeneration and trace monitoring in large C++ applications. Hylian uses an augmented form of the GNU *gcc* parser to output parse trees in XML format, permit modification of the parse trees, and to regenerate a modified version of the source code. We used an earlier version of Hylian to reverse engineer a grammar for the *gcc* C++ parser, version 4.0.0 [5]; the current version permits modification of the parse tree to extract trace information of a C++ application under study.

To demonstrate the utility of Hylian, we generate trace information, extract the sequence of method invocations, and generate regular expression representations of the interface protocols for *Necko*, a large networking library written in C++ [9]. To exercise the *Necko* library, we use three large applications in the Mozilla Internet Application Suite: Firefox, Thunderbird and Sunbird, a browser, mailer and calendar application respectively [8]. We then choose classes in the *Necko* library that are used by the three applications and examine the regular expression representation of the interface protocols for the *Necko* library.

Our preliminary results support our assumption that the interface protocols for these classes follow a specific pattern and that these patterns can be used to facilitate comprehension of the class and to guide usage of the class by developers unfamiliar with the *Necko* library. Moreover, the regular expression representations of the interface protocol for a class can serve as examples, or templates, of correct usage of a class for a large library. We conjecture that these examples of library usage exemplify the comprehension model needed in the maintenance of large libraries and, together with other comprehension tools, can facilitate the maintenance effort.

In the next section, we review the terminology and concepts that we use in our work. In Section 3 we describe our trace monitoring methodology and its use in reverse engineering interface protocols. In Section 4 we present the case study described above and in Section 5 we review related research. In Section 6 we draw conclusions.

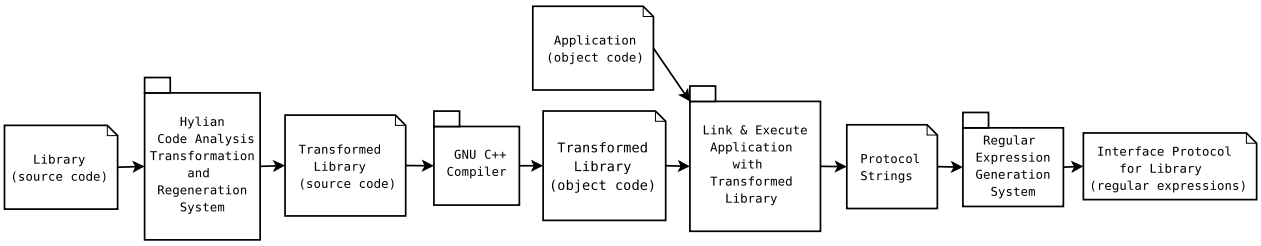


Figure 1. The Hylian **System.** *This figure illustrates* Hylian*, our code analysis and trace monitoring system.*

2. Terminology and Concepts

In this section, we review the terminology and concepts that we use in our work. In Section 2.1 we review grammars and parse trees, and in Section 2.2 we review the concept of *trace monitoring*.

2.1 Grammars, Parse Trees and ASGs

A grammar defines a language by specifying valid sequences of derivation steps that produce sequences of terminals, known as the *sentences* of the language. One procedure for using a grammar to derive a sentence in its language is to begin with the start symbol S and apply the production rules in some sequence until only non-terminals remain. This process defines a tree whose root is the start symbol, whose nodes are non-terminals and whose leaves are terminals. This tree is known as a *parse tree*; the process by which it is produced is known as *parsing*. Our system, Hylian, generates parse trees that we augment to monitor the execution of the libarary unders study.

2.2 Trace Monitoring

A *trace monitor* is a software artifact that observes the actions in a software system and, when certain activities are detected, the monitor executes some code of its own [2]. Trace monitors are especially useful for the detection or verification of runtime behavior. In our work, we use trace monitoring to detect class method invocations and to record a history of these invocations.

3. Protocol Extraction Methodology

In this section, we describe our system for monitoring the execution of an application and its corresponding library, and for reverse engineering interface protocols for C++ classes in the library. In the next section we present the *Hylian* system that we utilize and in Section 3.2 we describe our approach to regular expression generation.

3.1 Overview of the Hylian System

Figure 1 summarizes the flow of information through the system that we use. The source code for a C++ library is shown as input to Hylian, shown as a tabbed box to the left side of the figure. Hylian uses an augmented version of the GNU *gcc* parser, version 4.0.0, to generate a parse tree representation of the library code in XML format [5]. We produce transformed code to monitor the library by augmenting the parse trees with parse subtrees that contain code to trace the method invocations in the library; this phase is illustrated in Figure 1 as a tabbed box labeled Transformed Library. The Transformed Library is compiled into object code by the GNU C++ compiler, which is linked with the object code for the application that will utilize the library. The resulting executable, together with the input to the application, produces the Protocol Strings, which are then transformed into the Interface Protocol for the library, expressed as regular expressions.

3.2 Construction of Regular Expressions

We use an iterative algorithm to convert each protocol string into a regular expression and, for a protocol string of length n, our algorithm runs in $O(n^3)$ time. We first search the protocol string for recurring patterns of size 1, then recurring patterns of size 2, and continue the search, looking for recurring patterns of size $n/2$. For example, in searching for patterns of size 2, the string "abab" will be converted to $(ab)^+$. When the protocol string for each object is converted to a regular expression, we then use a *perl* package, Regexp::Assemble, to construct a single regular expression from the set of protocol strings generated by each instantiation of the class under consideration.

There is an abundance of research describing techniques to recover interface protocols using a finite state machine or regular expression representation [4, 7, 10, 11]. Our future work includes an investigation into these techniques to improve our protocol recovery process.

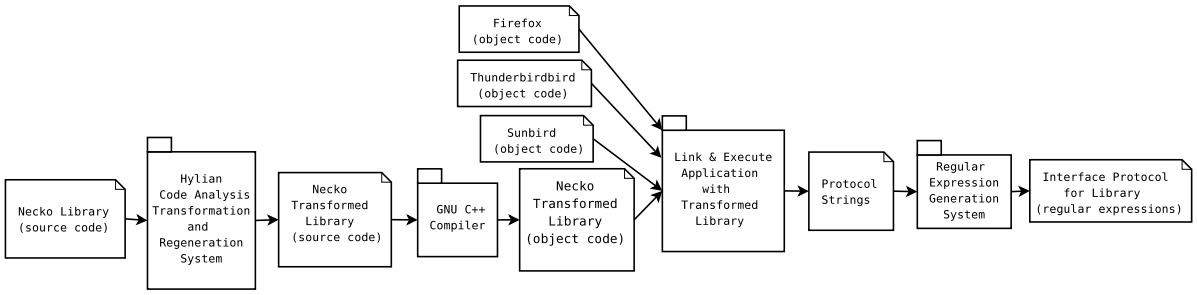


Figure 2. Study Summary. *This figure illustrates our study to evaluate our methodology for generating interface protocols for the Necko Networking Library utilized by the Mozilla Internet Application Suite.*

Application	Version	Units	Parse Tree
Necko	2.0a1pre	101	1,542,653
Firefox	3.0a8	1,262	27,857,127
Sunbird	0.6a1	1,586	33,023,605
Thunderbird	3.0a1pre	1,826	35,190,012

Table 1. Testsuite Statistics.

4. Case Study: The Mozilla Internet Suite

In previous sections of this paper, we described our approach for exploiting dynamic trace monitoring to capture the interface protocols of classes in an object-oriented system. In this section we describe a study that we conducted to evaluate our methodology and to show the utility of our approach. Our study involves an investigation into three applications included in the Mozilla Internet Application Suite: *Firefox*, a commonly used browser; *Thunderbird*, a mailer; and *Sunbird*, a calendar management application [8]. We use these applications to investigate usage of the *Necko* networking library included in the Mozilla Internet Application Suite.

Figure 2 illustrates our use of the Hylian analysis system, presented in Section 3, for transforming library code to provide dynamic trace monitoring and generate interface protocols for the classes in a library. In our study, we use the *Necko* library, listed on the left side of the figure, as input to our system, transform and regenerate the code, and use the GNU C++ compiler to generate object code for the transformed library, *Necko* Transformed Library, shown in the middle part of the figure. We then link the object code representation of the *Necko* Transformed Library, first with an object code representation of the *Firefox* application, and then with an object code representation of the *Thunderbird* application, and finally with an object code representation of the *Sunbird* application to produce protocol strings for each application. We then find the union of the protocol strings generated for *Necko* usage by each application to produce protocol strings for *Necko* classes, Protocol Strings, shown as a folded edge box (folded box) at the middle right of the figure. We then generate an Interface Protocol for Library, as regular expressions.

4.1 The Mozilla Application Suite

The statistics in Table 1 provide information about the version and size of the four applications that we study in this section. The first column lists the application, Application, the second column lists the version number, Version, the third column lists the number of compilation units, Units, and the fourth column lists the number of lines in the parse tree files, Parse Trees, for each application. The applications are ordered in the table by their parse tree size. For example *Necko*, the Mozilla networking library, is listed in the first row of the table and consists of version 2.0a1pre, 101 compilation units and 1,542,653 lines in the parse tree file. The largest application, *Thunderbird*, is listed on the last line of Table 1 and consists of version 3.0a1pre, 1,826 compilation units and 35,190,012 lines in the parse tree file.

4.2 Usage of the *Necko* Networking Library

Table 2 provides detailed information about the usage of *Necko* by the *Firefox*, *Thunderbird*, and *Sunbird* applications, including some information about the generated protocol strings for *Necko* classes. To exercise the *Firefox* browser, we visited medium-sized websites containing a number of images: *nytimes.com*, *washingtonpost.com*, *slashdot.org*, and *digg.com*. To exercise the *Thunderbird* application, we sent two emails consisting of plaintext and HTML markup, and we received an email message with a 1MB file attachment. To exercise the *Sunbird* calendar application, we synchronized the calendar component with three ICL formatted online calendars.

Application	Classes	Instantiations	Protocol Strings		
			Number	Longest String	Average Length
Firefox	76	14,055	14,055	59,070	31
Sunbird	55	1,297	1,297	17,578	68
Thunderbird	54	1,591	1,591	47,649	88

Table 2. Usage of *Necko* by the three Mozilla applications.

The first three columns of Table 2 list the application, Application, the classes used, Classes, and the number of classes instatiated, Instantiations, by the respective application. For example, the *Firefox* application used 76 of the 142 classes in *Necko*, which is 21 more classes than the *Sunbird* application used, and 22 more classes than the *Thunderbird* application used, even though *Firefox* is the smallest of the three applications, as measured by number of compilation units and number of lines of parse tree code (cf. Table 1). The sum of the instantiations in the third column (14,055+1,297+1,591) is 16,943, the total number of class instantiations for all three applications.

The final three columns in Table 2 list information about the protocol strings, Protocol Strings, generated by each of the three applications. The fourth column lists the number, Number, of strings generated and, since each object generates a protocol string, the number of strings is the same as the number of objects listed in the third column of the table. The fifth column lists the length of the longest string, Longest String, generated by each of the respective applications. The *Firefox* application generated the longest protocol string containing $59,070$ method invocations, which means that one of the instantiated classes made $59,070$ invocations of methods in the *Necko* library. The final column of Table 2 lists the average length, Average Length, of the protocol strings generated by the objects of the respective application.

4.3 Class Comprehension in Large Systems: Regular Expressions for Necko

Figure 3 contains two tables that describe information about the use of class nsDiskCacheInputStream in the Necko networking library. The table at the top of the figure contains four columns listing the name of the application, Application, the number of instantiations of nsDiskCacheInputStream, Instantiations, the number of unique sequences of method invocation strings for nsDiskCacheInputStream, Unique Sequences, and the interface protocol expressed as a regular expression, Interface Protocol. The first row of the table at the top of the figure lists information for *Firefox*, which created 166 instantiations of nsDiskCacheInputStream, generated 12 unique method call sequences that are summarized by the regular expression $abc^{+}ded$. The last two rows of the table list information for *Sunbird* and *Thuderbird*, which did not create any instantiations of class nsDiskCacheInputStream and did not use any of the methods.

The table at the bottom of Figure 3 has two columns where the first column, Mapping, specifies the mapping between letters and method names and the second column, Unique Sequences of Method Calls, lists the set of unique sequences of method invocations made by class instantiations of the *Firefox* application on the *Necko* networking library. For example, the first row of the second column of the table lists a sequence of 30 method invocations consisting of calls to ab, followed by a sequence of 25 calls to c, followed by calls to ded. Similarly, the ninth row of the second column of the table summarizes a sequence of 158 method invocations consisting of calls to ab, followed by a sequence of 153 calls to c, followed by calls to ded. Note that we use dots to indicate that some of the 153 calls to c have been elided from the ninth row of the table; however, all of the other sequences are illustrated precisely as they were generated by our test cases.

The regular expression representation of the 12 sequences is listed in the fourth column of the first row of the table at the top of Figure 3, $abc^{+}ded$. In lieu of documentation, UML case tool artifacts or other specification of the usage of a class, the reverse-engineered interface protocol can provide invaluable information about how a class in a large system is used, or may be used. For example, using the mapping, the typical usage of an instantiation of class nsDiskCacheInputStream consists of a call to the constructor, a call to AddRef, followed by one or more calls to Read, then calls to Close, Release and Close.

4.4 Comparison of Class Usage

Figure 4 illustrates a class instantiation history for those classes in the Necko library that were used by the Firefox, Thunderbird and Sunbird applications. Since we are interested in comparing usage by all three applications, the figure only lists those classes that were used more than ten times by each of the applications.

The three bars on the left side of the figure represent usage for the nsFileOutputStream class in the Necko library where the first bar indicates that Firefox created 44 in-

Use of nsDiskCacheInputStream by Three Applications			
Application	Instantiations	Unique Sequences	Interface Protocol
Firefox	166	12	$abc^{+}ded$
Sunbird	0	0	NA
Thunderbird	0	0	NA

Protocol Strings for *Firefox* Application	
Mapping	Unique Sequences of Method Calls
a → nsDiskCacheInputStream b → AddRef c → Read d → Close e → Release	01: abccccccccccccccccccccccded 02: abccccccccccded 03: abcccccccccded 04: abccccccccccccded 05: abcccded 06: abccccccccccccccded 07: abcccccded 08: abcccccccded 09: abccccccccccccccccccccccc . . . ccccccded 10: abccccded 11: abccded 12: abcccccccded

Figure 3. Method Invocation Sequences for Class nsDiskCacheInputStream**.**

stances, Thunderbird created 37 instances and Sunbird create 28 instances of this class. The three bars in the middle of the figure represent usage for the nsBufferedStream class in the necko library where the first bar indicates that Firefox created 66 instances, Thunderbird created 68 instances and Sunbird create 56 instances of this class. The usage of this class is more evenly distributed than the usage of nsDiskCacheInputStream illustrated in Figure 3.

We found that the generated regular expressions for more heavily used classes in Necko can be less readable than the regular expression, $abc^{+}ded$ that we obtained for class nsDiskCacheInputStream. For example, Firefox usage of class nsDiskCacheInputStream generated the expression $abbcdbcb^{*}fbg^{*}ch^{*}(ggh)^{*}c^{*}e^{*}$, which may not be as useful as the one we obtained for class nsDiskCacheInputStream, or may indicate that nsDiskCacheInputStream has more complex usage patterns. Our ongoing work includes an investigation of some of the excellent regular expression generation algorithms in literature [4, 7, 10, 11].

5. Related Work

The generation of interface protocols can be accomplished using either static or dynamic analysis. The static approach has the advantage of finding all possible sequences of method invocations but must address the problems of pointer alias [6] and infeasible paths. Moreover, the static approach may provide interface protocols that are irrelevant to the application under consideration, as we have seen in Section 4.4. the regular expressions can be overly complicated. The dynamic approach has the advantage of providing only those sequences of method invocations that are relevant to the application under consideration and does not suffer the problems of alias or infeasible path resolution. The related work that we review in this section employs the dynamic approach to protocol recovery.

Cornelissen and Moonen describe a technique for addressing the scalability problem in extracting information from execution traces of function calls in Java programs [4]. They observe that certain event sequences are repetitive, where the repitition typically results from the occurrence of method invocations within loops. Their summarization technique entails the use of *similarity matrices* to visualize the repetitive method invocation sequences in the trace. Recurring sequences of method invocations appear as patterns in the matrix. However, the sequences appear in the matrix as diagonal lines and this abstraction results in the loss of information about the identity of the methods in the recurring sequences. The approach that we describe in this paper entails summarizing the recurring sequences of method invocations as regular expressions, which has the advantage of maintaining the identity of the methods involved in the recurring sequences.

Walkinshaw et al. describe the construction of state machines from user supplied scenarios and execution traces of Java programs [10]. They use the scenarios from the user, the execution traces and the QSM state-merging approach to interactively generate a state machine of the system. Our approach differs from that of Walkinshaw et al. in that our technique is fully automated and does not require user supplied scenarios.

Butkevich et al. describe an extension to the Java programming language to facilitate static conformance checking and dynamic debugging of object protocols [3]. *Object protocols* are sequencing constraints on the order in which methods in a Java application may be invoked. In their

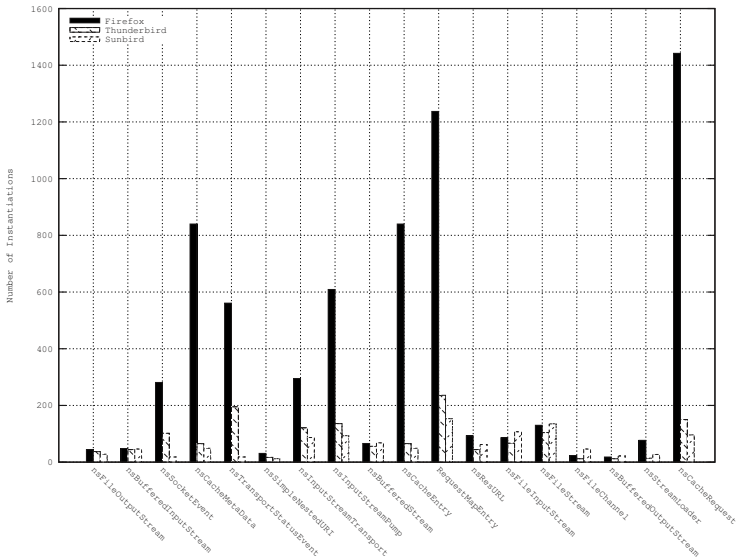


Figure 4. Necko Usage by Firefox, Thunderbird and Sunbird.

work, regular expressions are used to specify the conformance relation between two object protocols. However, the work of Butkevich et al. does not entail the reverse engineering of existing interface protocols, nor do they apply usage of interface protocols to program comprehension.

Archer et al. develop a checkable, executable specification that captures the rules for correctly using an interface in a TinyOS application [1]. They refer to this specification as an *interface contract* and they develop an approach for checking the interface contract using a source-to-source program transformation that adds checks to existing TinyOS applications. However, they do not reverse engineer the contracts and they do not demonstrate the pattern that these contracts typically exemplify.

Quante and Koschke describe a dynamic protocol recovery technique based on *object process graphs* (OPGs) [7]. The advantage of their approach is that OPGs contain information about loops and the context within which a method was called. They introduce a new metric for comparing automata and a case study involving Java and C programs. The focus of Quante and Koschke is on their recovery protocol technique that exploits context to improve the regular expression generation process; their technique improves on our brute force regular expression generation approach.

6. Concluding Remarks

We presented Hylian, our system for code regeneration and trace extraction in large C++ applications. We have demonstrated the utility of Hylian by generating trace information, extracting the sequence of method invocations, and generating regular expression representations of the interface protocols for *Necko*, a large networking library written in C++, utilized by the Mozilla Internet Application Suite [9]. We generated interface protocols for *Necko* using three large applications in the Mozilla Suite that use the *Necko* library: Firefox, Thunderbird and Sunbird [8].

The preliminary results of our case study of *Necko* support our assumption that the interface protocols for these classes follow a specific pattern and that these patterns can be used to facilitate comprehension of a class, to guide usage of the class, or to measure the complexity of usage of the class by developers unfamiliar with the *Necko* library.

References

[1] W. Archer, P. Levis, and J. Regehr. Interface contracts for tinyos. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 158–165, New York, NY, USA, 2007. ACM Press.

[2] P. Avgustinov, J. Tibble, E. Bodden, O. Lhotak, L. Hendren, O. de Moor, N. Ongkingco, and G. Sittampalam. Efficient trace monitoring, March 2006.

[3] S. Butkevich, M. Renedo, G. Baumgartner, and M. Young. Compiler and tool support for debugging object protocols. In *Proceedings of the 8th FSE*, pages 50–59, 2000.

[4] B. Cornelissen and L. Moonen. Visualizing similarities in execution traces. In *Proceedings of the 3rd Workshop PCODA*, pages 6–10. IEEE, 2007.

[5] E. B. Duffy and B. A. Malloy. An automated approach to grammar recovery for a dialect of the C++ language. In *14th Conference on Reverse Engineering, WCRE'07*, Oct 2007.

[6] D. Engler, B. Chelf, A. Chou, and S. Hallem. Checking system rules using system-specific, programmer-written compiler extensions. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, 2000.

[7] J. Quante and R. Koschke. Dynamic protocol recovery. In *Proceedings of the 14th WCRE*, pages 219–228, Washington, DC, USA, 2007. IEEE Computer Society.

[8] The Mozilla Organization. The Mozilla Application Suite. *http://www.mozilla.org*, 2007.

[9] The Mozilla Organization, 2007. Necko. *http://www.mozilla.org/start/1.0/guide/toolkit.html#necko*.

[10] N. Walkinshaw, K. Bogdanov, M. Holcombe, and S. Salahuddin. Reverse engineering state machines by interactive grammar inference. In *Proceedings of WCRE '07*, pages 209–218, Washington, DC, 2007.

[11] J. Whaley, M. C. Martin, and M. S. Lam. Automatic extraction of object-oriented component interfaces. In *Proceedings of ISSTA*, July 2002.

Knowledge Management to Support the Deployment of a CMMI Level 3 Process

Cavalcanti A. P.[1,2], Furtado F.[1,2], Moura V. [1], Costa, R. [1,2], Meira, S. R. L. [1,2]

[1.] C.E.S.A.R – Recife Center for Advanced Studies and Systems
[2] UPFE – Federal University of Pernambuco, Cin – Informatics Center
{ana.paula, felipe.furtado, valeria.moura, ricardo.costa, silvio}@cesar.org.br

***Abstract*—Knowledge Management area has gathered great interest in software development industry and the applicability of its process into an organization, In order to support the requirements of a maturity level, is the main focus of this report. Thus, CMMI requirements were assessed and introduced into a KM project that was run into a CMMI level 3 organization and the results were analyzed to evaluate whether or not it was useful to apply KM into such scope.**

***Index Terms* — CMMI, Knowledge Management, Software Development Process.**

I. Introduction

KNOWLEDGE management discipline is gathering a growing attention on organizations that seek the establishment and tracking of the knowledge disseminated throughout the company. Procedures, processes, tools, among other initiatives are supporting the use and dissemination of such discipline within software development context.

Therefore the objective of this work is to present the knowledge management approach used to support the deployment of a CMMI level 3 process into a software development organization. In addition, to assess the results achieved by the use of such process in order to provide improvement opportunities to the organization.

This report is organized as it follows: Section II presents a general overview on Knowledge Management and the basic concepts used to develop the process, followed by the research of where KM requirements are found on CMMI [1] on Section III. Section IV describes the process approach defined by the organization to adopt KM, Section V details the tool adopted and the reasons that show why such tool is being used, and, Section VI presents the case study results. Finally, Section VII shows the concluding remarks and future work.

II. Knowledge Management Overview

For the last years, the Knowledge Management area is receiving a special attention from organizations, as they search for competitive advantage. The following statics are a simple proof of this attention [2]:

1) A total of 80 percent of Fortune 500 companies have KM staff.
2) Texas Instruments has saved $1 billion since it launched KM programs in the mid-1990s.
3) 95 percent of CEOs polled at the 2001 World Economic Forum in Davos, Switzerland, said that KM was critical to organizational success.
4) 91 percent of Canadian business leaders polled by IpsosReid in 2001, believed that KM practices have a direct impact on organizational effectiveness.

In [3], Polanyi categorized knowledge in two types, the tacit and explicit. The latter is basically what can be documented and distributed, while the tacit knowledge resides in the human mind, behavior and perception, and thus, it is difficult to be formalized and distributed [4].

Each KM program needs to balance which kind of knowledge it is focusing, and based on this focus it can be categorized in one of the four KM styles defined by Choi and Lee [4].

1) Passive: little interest in KM. It is not managed in a systematic manner.
2) System-oriented: put more emphasis on codifying and reusing knowledge. It increases codifiability through IT, and thus, decreases the complexity of accessing and using knowledge.
3) Human-oriented: the emphasis is on acquiring and sharing tacit knowledge and interpersonal experience. Knowledge usually originates from informal social networks. Meaningful knowledge can not be simply retrieved form the database or repository.
4) Dynamic: emphasizes both tacit and explicit knowledge, and does it in a dynamic fashion, similar to a communication-intensive organization. They depend on cultural knowledge.

Independent of style, knowledge management can be seen as a process to create, gather, store, transfer and apply knowledge [5]. There are many different definitions for the stages composing a KM process. In 2004, Bose presented the stages which compose a cyclic process in knowledge management

[2]:

1) Create knowledge: the knowledge comes primarily from the experiences and skills of the employees. Knowledge is created as people find new ways to do things. Sometimes it is necessary to brought in external knowledge when it does no reside in the organization.
2) Capture knowledge: the created knowledge is stored in a database or repository in raw form.
3) Refine knowledge: context is added to knowledge, so it can be easily reused. In this stage the tacit knowledge, usually captured from human experience, is captured and transformed and refined along with explicit knowledge.
4) Store knowledge: This stage includes the codification of tacit and explicit knowledge, after refined, so it can be used later.
5) Manage knowledge: knowledge must be kept current, so it must be reviewed systematically to verify if it is still valid and accurate;
6) Disseminate knowledge: The knowledge is made available so everyone in the organization can easily access it anywhere and anytime. New technologies are usually used to help in the dissemination of knowledge.

Some of the several benefits of the use of a knowledge management strategy are: the reduction in loss of intellectual capital from employees who leave the company; the cost reduction for the development of new products; and the increased productivity by making knowledge easily accessible to all employees [2].

III. KNOWLEDGE MANAGEMENT AND CMMI

Knowledge management process definition was being established together with the adaptation of the organizational process from CMM level 2 into CMMI level 3. Therefore, it was necessary to understand which requirements, from the CMMI model [1], treat knowledge management general aspects.

Accordingly, CMMI level 3 has three process areas (PAs) that address KM's necessities. Table I presents the mapping among these PAs together with a short description.

Based on the sub-practices from CMMI, OPF and OPD process area define the organizational approach to posterior use on projects, as for instance:

- "Conduct periodic reviews of the effectiveness and suitability of the organization's set of standard processes and related organizational process assets relative to the organization's business objectives.
- Obtain feedback about the use of the organizational process assets.
- Derive lessons learned from defining, piloting, implementing, and deploying the organizational process assets.
- Make lessons learned available to the people in the organization as appropriate.
- Design and implement the organization's process asset library, including the library structure and support environment; to specify the criteria for including items in the library.
- Specify the procedures for storing and retrieving items; to enter the selected items into the library and catalog them for easy reference and retrieval.
- Make the items available for use by the projects.
- Periodically review the use of each item and use the results to maintain the library contents.
- Review the organization's process asset library as necessary."

TABLE I
MAPPING BETWEEN CMMI AND KM

Process Area (PA)	Specific Goal (SG)	Specific Practice (SP)	Description
OPF Organization Process Focus	SG 1 Determine Process-Improvement Opportunities	SP 1.3-1 Identify the Organization's Process Improvements	Identify improvements to the organization's processes and process assets.
	SG 2 Plan and Implement Process Improvement Activities	SP 2.4-1 Incorporate Process-Related Experience into the Organizational Process Assets	Incorporate process-related work products, measures, and improvement information derived from planning and performing the process into the organizational process assets.
OPD Organization Process Definition	SG 1 Established Organizational Process Assets	SP 1.5-1 Establish the Organization's Process Asset Library	Establish and maintain the organization's process asset library, like procedures and lessons-learned reports.
IPM Integrated Project Management	SG 1 Use the Project's Defined Process	SP 1.2-1 Use Organizational Process Assets for Planning Project Activities	Use the organizational process assets and measurement repository for estimating and planning the project's activities.
		SP 1.5-1 Contribute to the Organizational Process Assets	Contribute work products, measures, and documented experiences to the organizational process assets. Document lessons learned from the project for inclusion in the organization's process assets library.

The other process area, IPM, is concerned about how the projects will use the organizational process assets for planning new project activities and then how the projects will contribute to the organizational process assets, for example:

- "Use the organization's measurement repository in estimating the project's planning parameters.

- Propose improvements to the organizational process assets.
- Store process and product measures in the organization's measurement repository.
- Submit documentation for possible inclusion in the organization's process asset library.
- Document lessons learned from the project for inclusion in the organization's process asset library."

Based on this research, these requirements were used as a basis to build the KM process, which will be described on the next section.

IV. PROSCES AND KM

According to the characteristics presented on the introduction, the process that will be described intends to *Dynamically Manage* the knowledge acquired, emphasizing both tacit and explicit. Furthermore, the organizational approach to implement a formal knowledge management process is formed by a process definition, a support tool, process templates and guidelines. Such KM process, called *ProSCes*[1], within the context of software development, is considered a *support sub-process*, due to the fact that it permeates the whole software development cycle, and the main purposes are:

- Stimulate the change of knowledge between professionals through the record and use of information in the knowledge organization base.
- Identify and organize lessons learned, that represent the knowledge applied to the projects reality.
- Provide the spread of global knowledge, available to the entire organization areas.

This process can be used by people from all levels of the software engineering activities, such as Project Managers, System Engineers, Quality Engineers, Configuration Management Engineers, Human Resource, Operations Manager, Quality Manager, Project Manager Officer and Engineering Manager.

An important relevant factor is that ProSCes differentiates *knowledge* and *Lessons Learned (LL)*, where the first one is any knowledge acquired by the organization collaborators derived from professional experience, studies and research. While, on the other hand, *Lessons Learned* are the relevant practical results acquired from real situations in projects specifically at C.E.S.A.R.[2].

The process is organized into 6 main activities:

A. *Disseminate the identification and use of knowledge into projects*

This activity focuses on the identification and registry of existent knowledge, promote continuous and effective use of the knowledge available and stimulate the collection of lessons learned in the projects. It maintains discussion forums about organizational interests and stimulates changes in organizational knowledge.

The main outputs of this activity are lessons learned registered on Mantis (the tool that supports the process and which will be explained in detail on the next section) and knowledge registered either on C.E.S.A.R Wiki[3] or A.M.I.G.O.S[4].

B. *Plan lessons learned collection*

The main objective of this activity is to plan the collection of lessons learned by the identification of specific milestones in the scope of project or in the scope of the organization. As outputs, the Software Quality Engineer (SQE) updates the project plan with the milestones of project and Software Engineer Process Group (SEPG) updates the organizational plan to contemplate the collection lessons learned in the organization.

C. *Execute lessons learned*

This activity focuses on the identification of lessons learned in the project/organization. In the scope of the project, SQE configures a questionnaire based on the project milestone and, in the scope of the organization SEPG configures it based on which sub-processes will be assessed. The teams answer the questions based on the knowledge acquire and theses answers are collected and consolidated to be presented by each specific audience (project or organizational).

As a result, a report is generated with all the knowledge consolidated so that the SQE or SEPG selects which of the knowledge can be considered lessons learned. Another possible output are change requests opened for the software development processes, that will be requested by SQE or SEPG to the organization process group.

D. *Maintain best documents and lessons learned data*

The purpose of the activity is to register the selected lessons learned on Mantis and maintain the information available, so that teams have direct access to the information. The activity steps start with the registration of the lessons learned on Mantis, followed by the facilitation of the lessons learned access for the whole organization, and the maintenance of the Mantis repository with actual and relevant information for all organizational projects.

SQE and SEPG are responsible for the lessons learned registry in the repository, which are registered according to a set of pre-defined parameters that will be described in details on the Section V, which talks about Mantis tool.

This activity generates the register of lessons learned, best documents, recommended and non-recommended practices in the repository.

E. *Disseminate lessons learned usage*

The proposal of this activity is to encourage the use of projects lessons learned into the organizational context and

[1] **ProSCes**: Software Process of C.E.S.A.R, available at the intranet of the organization, not accessible for outsiders. http://cesar.org.br/prosces

[2] **C.E.S.A.R**. – Recife Center for Advanced Studies and Systems. http://www.cesar.org.br

[3] **C.E.S.A.R Wiki,** Available at the intranet of the organization, not accessible for outsiders. http://cesar.org.br/wiki

[4] **A.M.I.G.O.S** – Multimedia Environment to the Integration of Groups and Social Organizations. http://amigos.cesar.org.br

provide the exchange of experiences among projects.

The responsible for this activity is SEPG, that should promote seminars presenting the content generated the knowledge repository. On the other hand, SQEs should encourage team for the continuous use of the lessons learned repository.

F. Identify process improvement opportunities

This activity focuses on the identification of process improvement opportunities based on the learned generated by the project in the organization.

The SEPG periodically analyzes the lessons learned repository to identify opportunities to improve the organizational process. Typically, lessons learned that are repeated in various projects and lessons learned about a specific sub-process activity are examples such opportunities. Modifications to the process can happen in terms of an inclusion of activity, update an existing on, inclusion of new process techniques, guides elaboration, among other aspects.

V. KM APPLIED TOOL

In order to support the use of knowledge management into an organization, it is necessary to set up an environment that can contribute to the use and improvement of knowledge management processes. In order to choose which environment could be best applied to our context, a formal decision taken process, described as one of the sub-processes available at ProSCes, was run in order to evaluate the existing solutions available.

The first part of this process contemplates the identification of criteria and weighs to evaluate the possible tools, according to what is described on Table II.

TABLE II
TOOLS' CRITERIA

Criteria	Weigh
Integration with the organizational projects base	20
Agility to implement the solution	15
Agility to find the lessons learned	25
Performance to include information (Lessons Learned)	10
Technical risk of the alternative that may exist to implement the solution	30

These criteria were used to analyze three solutions to support KM deployment into the organization:

1) **Mantis**: the implementation of lessons learned repository through the use of mantis tool, which "is a popular free web-based bug tracking system" [6].
2) **Rational Portfolio Manager – RPM**: the implementation of lessons learned repository through the use of RPM, which is a tool to support project integrated management [7]
3) **Mantis with integration**: the implementation of lessons learned repository through the use of mantis tool with an integration with RPM data.

The conclusion of the process led to the results presented in the graphic on Figure 1.

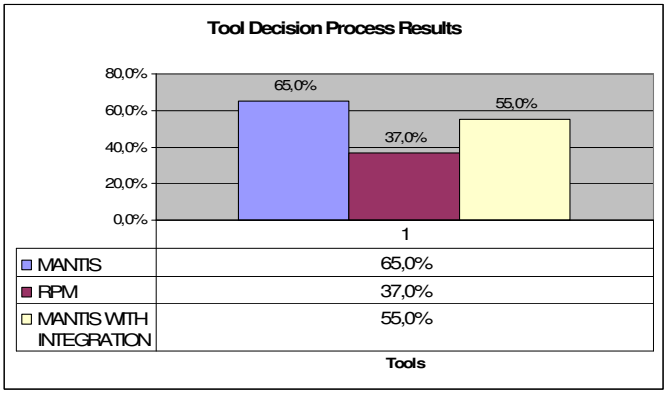


Fig. 1. The results achieved through the application of a formal decision process to decide which tool would best fit the knowledge management process applied by ProSCes.

The results of the decision taken process calculate the percentage of criteria (described on Table II) satisfaction for each tool identified, and, consequently, their ranking in the process. Therefore, 65% of the criteria are satisfied by Mantis tool, against 37% for RPM and 55% for Mantis with integration with RPM.

Based on these considerations, ProSCes adopted Mantis, and although mostly used for a bug tracking system, it can be easy configured in order to achieve desirable levels of usage to a knowledge management support tool. Mantis internal structure was adapted to contemplate the following arrangement, where a lesson learned can be described according to the following parameters:

Categories

- **Best Document**: documents that are considered a model of reference to the organization in a determined sub-process, for example, a requirements specification document or an architectural design document.
- **Recommended Practice**: experiences acquired from the process execution that are considered successful and which could be reproduced in other projects.
- **Non-recommended Practice**: experiences lived by projects team that are not considered successful and should not be reproduced in other project with the same scope.
- **Feedback**: from the team related to the processes releases, regarding whether the change was useful or not for the project and the proposal of changes to the organizational process.

Moment of Collection

- **Process Audit**: during a formal process audit conducted based on a specific sub-process area, where the SQE tries to identify lessons learned gained by the projects.
- **Metrics Analysis**: where the team consolidates process indicators and points out any process improvement or lessons learned from the metrics

analysis.

- **Execution of Lessons Learned**: a formal collection of lessons learned conducted by the SQE through the use of questionnaires applied on specific project teams, where there is a report as the activity output.
- **Other**: any other moment during process development.

Source of Collection

- **Own Initiative**: the source that comprehends to knowledge that comes from individual initiative, that is not formally registered anywhere.
- **Knowledge Repository**: database where all collaborators have free access to input any knowledge, like C.E.S.A.R. Wiki or A.M.I.G.O.S.
- **Lessons Learned Repository**: is related to Mantis tool itself, where lessons learned can be raised from other lessons learned previously registered.

Transitional States

- **Submitted**: the SQE submits the lesson learned to Mantis but it has to be reviewed by SEPG in order to publish it.
- **Published**: a lesson learned that has been published and reviewed by SEPG in order to guarantee that the content can be published to the organization.
- **Rejected**: a lesson learned that has been reviewed by SEPG and is not considered a lesson learned to be published to the organization.

Besides theses characteristics, mantis permits the creation of correlations among all lessons learned submitted, by the settlement of relationship, which can be: (1) one is *parent of* the other; (2) one is *son of* the other; (3) one is *duplicated with* another, and (4) one is *related of* another. These relationships are represented through the use of relationship trees, a graphical and visual representation of the lessons learned available on the knowledge database.

VI. Case Study

A. Scenario and Objectives

The case study was conducted with the main objective to assess the adoption of knowledge management through the use of ProSCes, together with Mantis. The institution applied is C.E.S.A.R., a seven hundred employee company located on the northeast of Brazil which, by the time of the process definition, was on the course of the CMMI level 3 assessment.

C.E.S.A.R.'s software development process considers many technologies (J2ME, J2EE, .NET, etc), distinct lifecycles and project categories, such as embedded software development, mobile applications, client-server applications, mobile games, test automation, among others. For the CMMI level 3 assessment, 4 projects were chosen as the most representative ones in the organization, and approximately 68 people were directly involved in the process to define and institutionalize the CMMI level 3 process.

Accordingly, this case study attempts to present the results of the KM Process execution since the definition of the process, following by the checkpoint of the CMMI appraisal, until the present moment.

The main motivation to asses how KM was introduced into ProSCes are:

- Analyze the results of the process to propose improvements.
- Analyze the use of Mantis as a knowledge management support tool.
- Analyze the involvement of the organization into the knowledge management process.
- Identify the main source of lessons learned within the development process.

B. Collected Data

Based on that scenario, some metrics were collected to support the case study, and the first on is presented on Table III. The numbers show that until the CMMI appraisal, most of the lessons learned were published, and after that period, most of them were just submitted.

TABLE III
LESSONS LEARNED DISTRIBUTION ALONG THE TIME

Lessons Learned	Before CMMI Appraisal From January 2007 until November 1st 2007 (%)	After CMMI appraisal From November 2nd 2007 until March 2008 (%)
Submitted	23 %	62 %
Published	47 %	30%
Rejected	30 %	7%

To better understand the involvement of the organization into the process, we analyzed the moment in which lessons learned were identified. The results show that, from the entire database:

- 30% of the lessons learned reported come from the activity *Execute Lessons Learned*.
- 44% of the lessons learned reported come from the activity to *Realize Process Audit*.
- 4% of the lessons learned reported come from the activity *Analyze Metrics*.
- 22% come from any other activity in the project.

It was also important to evaluate the percentage of distribution of the lessons learned categories, where they are divided into:

- 33% are *Best Document*
- 59% are *Recommended Practice*.
- 4% are *Non-recommended Practice*.
- 4% are *Feedback.*

Besides that, another relevant aspect is that 82% of the lessons learned available appear with *Own Initiative* as the source of information, against 12% of the others (6% *Knowledge Repository* and *6% Lessons Learned Repository*).

Another important number to evaluate is related to the submission of changes request opened in the bug tracking system for the organizational process, and only 2% are related

to the Knowledge management sub-process.

C. Results Analysis

According to the numbers collected, the characteristics which presented the greater percentage are summed up on Table IV.

TABLE IV
CHARACTERISTIC AND PERCENTAGE ACHIEVED

Characteristic	Percentage
Process Audit	44%
Recommended Practices	59%
Own Initiative	82%

By analyzing this Table we can conclude that *Process Audit* is the moments in which lessons learned were mostly collected, a moment when the SQE directly asks the team about any lessons learned. This indicates the other moments to collect LA are not being well explored either because of the process or the teams are not having enough maturity to identify Lessons Learned on those moments.

Recommended practices are the ones that have the greater appearance on Mantis database, and we can infer that the maturity of the organization tends to the discovery of practices that can be replicated on other projects. Non-recommended practices and feedbacks are not being frequently reported.

Another consideration is that 82% of the lessons learned reported are from the source *Own Initiative* which shows that the other source of information are not being effectively used. It is necessary to analyze whether the other repositories are necessary to the KM process, or are not suitable to its goals.

The number of change requests opened for the knowledge management sub-process, either for a process improvement or for adaptation represents only 2%, which does not mean that it is not relevant. This is due to the fact that we have 23 other sub-process that change requests can be opened.

Another representative analysis is based on the numbers of Table III, where we see that less lessons learned are being published after CMMI appraisal. This situation reflects the idea that SEPG has changed its priorities after the formal appraisal.

D. Improvement Opportunities

Based on the results analyzed, some improvement opportunities were raised to request changes on knowledge management sub-process, according to what is presented:

- Improve the process of lessons learned publication, by assessing if there can be added more agility to this process, in order not to be so dependant on the SEPG.
- Review Mantis tool in order to detail what is being effective, in terms of the existing categories, access to the submission of lessons learned, and collect an opinion of the organization about it through a formal research.
- Evaluate all source of information available on ProSCes and question whether or not all these disseminated knowledge and lessons learned could be integrated into one tool.

VII. CONCLUSION AND FUTURE WORK

The deployment of knowledge management process into a software development organization, during the process of improving a maturity level, has presented considerable results indicating positive remarks to the organization. By trying to achieve tacit and explicit knowledge, the process is improving its definition and applicability into projects.

On the other hand, the results presented on the case study induces us to the idea that there is a long path to run, in terms of process definition and tools to support and get the organization's involvement.

According to the stages presented by Bose [2], with the process definition and execution, we intend to follow all phases from the cycle process:

- Create: with the definition of the process.
- Capture: by the execution of the process.
- Refine: through process improvement change requests, and through new projects, with new contexts, using and registering its experiences with already existing knowledge.
- Store: with the use of mantis tool.
- Manage: together with SEPG that helps the maintenance of the process and database.
- Disseminate knowledge: which is a consequence of the process and their execution.

REFERENCES

[1] CMMI-DEV, CMMI for Development, V1.2 model, CMU/SEI-2006-TR-008. Software Engineering Institute, 2006.

[2] R. Bose, "Knowledge management metrics". *Industrial Management & Data Systems*, vol. 104, no. 6, pp. 457-468, 2004.

[3] M. Polanyi, "The tacit dimension," in: *Knowledge in Organizations*, Ed. London: Butterworths, 1997, pp. 135–146.

[4] B. Choi, H. Lee, "An Empirical Investigation of KM Styles and their Effect on Corporate Performance". *Information & Management*, vol. 40, pp. 403-417, 2003.

[5] K. C. Laudon, J. P. Laudon, "Managing Knowledge for the Digital Firm," in *Management Information Systems: Managing the Digital Firm*, 8th ed., Ed. Pearson Education, 2004, ch. 10.

[6] Mantis available at http://www.mantisbt.org

[7] RPM– Rational Portfolio Management Tool, client version 7.1.1.1 Build 6.7.1.226 ©

Code Transformation Techniques and Management Architecture for Self-manageable Distributed Applications

M. Muztaba Fuad
Department of Computer Science
Winston-Salem State University
Winston-Salem, NC 27110, USA
E-mail: fuadmo@wssu.edu

Abstract

Injecting autonomous behaviors into existing non-autonomous programs is desirable but requires a high degree of code transformation. Such code transformation is complicated and requires different level of code insertion and transformation to provide the required functionality. Proper underlying software architecture is required for managing the code once it is transformed into autonomous entities. All this is even become complicated when existing programs (whose source code is no longer available) want to utilize such code transformation and service architecture. This paper discusses techniques to transformation existing code into autonomous entities and presents underlying management architecture to run such transformed application.

1. Introduction

Programming a distributed application is a tedious task and programmers need expert knowledge of handling the distribution related management issues along with programming the problem at hand. This becomes even daunting when programmers have to incorporate autonomic behaviors into the system also. The scenario gets even complicated when developers want to transform existing non-autonomic distributed system (whose source code is no longer available) into an autonomic system. In real life, programmers' want to concentrate on the problem in hand, rather than spend time on incorporating autonomic behaviors in their system or deal with the distribution management issues. It will be tremendously beneficial to programmers if such autonomic behaviors can be added automatically and transparently to existing systems and distribution related issues can be addressed without spending much time for it. Although object oriented technology provides programmers with advantage of rapid program development in a networked environment, it becomes overwhelming when it is necessary to handle the autonomic computing aspects of the program as well. Towards this goal, the initial phase would be to how to convert an existing object oriented program into self-manageable chunks and access them autonomously across the system so that autonomic behaviors can be added into those autonomous entities as needed. This paper presents such code transformation techniques to produce autonomously accessible program component from existing object oriented programs and identifies issues related to such code transformations. Although the technique presented in this paper works with Java byte code, any interpreted code such as C#, that utilizes standard object oriented primitives can utilize it with minor modifications.

2. Related Works

There are several code transformation techniques in Java actively researched by a number of researchers [1-10] in different aspects of software development. However, these approaches either work towards traditional hard coded method or have different goals than this work:

- Although Addistant [1] works with Java bytecode, it uses a separate user specified placement policy to translate user code to a distributed version and requires placement policies be specified at the class level, limiting the opportunity to exploit object-level control. J-Orchestra [2] is similar research work that rewrites user code and replaces local data exchange with remote communication.
- Operate at the source code level [3].
- Program through pre-defined interfaces to utilize such approaches [4].
- Use monitoring and profiling during run time [5, 6].
- Use a modified Java Virtual Machine [7, 8], and sacrifices portability and interoperability for doing so.
- Use different programming languages [9, 10].

Please see [11] for detailed description and contrast of these works (most of the related work is not mentioned here for space constraints) with the work presented in this paper.

3. Code Transformations

After the user presented the code (computationally intensive, large parallel applications) to the system, a static code analyzer builds an object graph from the user supplied byte code. See reference [12] for more details on the static code analyzer. Once it generates the object graph, the graph is partitioned [13] according to the underlying system configuration, communication requirements or any user supplied policy. The underlying system comprises a collection of platform-agnostic autonomic elements [14] as an interface to the service providers and the associated pre-processor for comprehensive byte code to byte code translation, so that the resultant transformation produces a self-adaptive version of the user code. The transformed program is based on self-contained concurrent objects communicating through standard object based communication protocols and incorporates salient features (such as

Broker architecture and asynchronous call) from existing middleware technologies.

3.1 Autonomic transformations

Since the transformed program runs as a distributed program, any single node in the system could have one or more objects executing on it. As all of the related objects within any single node have to be manipulated transparently, several different scenarios present themselves for consideration:

a. *One AE(Autonomic Element) per node with each object as a ME (Managed Element* [14]*)*: The traditional proxy approach [1] could be employed; however, the full computation power of a particular node is not utilized in this scenario. The traditional proxy approach only supports a single object per proxy, but multiple objects per proxy are required for the approach presented in this paper.
b. *One AE per node with multiple objects as ME*: This is the most common scenario where multiple objects run concurrently in one node. The traditional proxy notation must be extended to address this scenario.
c. *Multiple AEs per node with multiple objects as a ME*: Although this scenario is possible, it is not supported by the autonomic computing paradigm and is not considered further.

A proxy structure is implemented that encapsulates one or more runtime objects into a single manageable entity that communicates with other such entities in the system with a single communication channel provided by the proxy object. Having an encapsulating proxy object allows us to incorporate the autonomic functionalities seamlessly into the user objects with the help of sensors, actuators and control interfaces. Any inter-object communication inside a single ME proceeds as usual; however, any inter-object communication between two different MEs is delegated to the encapsulating proxy object. There are two possible choices to create such an encapsulating proxy:

a. *The proxy class inherits the original class*: This works for only one class as Java does not support multiple inheritance. This can be overcome by creating a new interface with all the methods of each of the target classes and then having the proxy implement that interface by copying the method's body into the proxy class.
b. *Renaming the original class with the proxy class*: This is the traditional approach and does not work without modification for the proposed approach as it is difficult to delegate all proxy invocations separately. This approach needs to be extended by creating a clone of the original class structure as the new proxy structure and, at the byte code level, redirect all calls to the proxy class. This allows the existing methods to be overridden with additional functionalities and the class to be extended with new methods. Since all associated transformations are performed at the byte code level, users do not need to be concerned about following any specific programming rules.

For such transformations of user code, the following issues need to be addressed:

i. *Methods (M) and constructors (C)*: There are more methods in the proxy class to interact with the AE and also to manipulate the object itself. So, if M_n , $C_n \in$ *Original Class* $\rightarrow$ $M_{n'}$, $C_{n'} \in$ *Proxy Class,* where $n' > n$. The original methods are overridden with the following structure:

Pre-processing
Original method call
Post-processing

Instead of instrumenting each method, a wrapper method is created to ensure consistency with the existing line number table for debugging purpose.
ii. *Polymorphic method calls*: To determine the original calls of a method in a polymorphic call requires a stack oriented emulator such as that in the JVM. Creating such an emulator is a separate research problem and in the initial version of the transformations, polymorphic method calls are not considered.
iii. *Direct field access*: All direct field access is converted to getter and setter methods to facilitate remote method invocation.
iv. *System classes*: Since the system classes cannot be modified, the same techniques as used in J-Orchestra [2] are adopted. System objects are either moved with the user objects or, if they use any platform dependent code, remain on the same node and other proxies access these system objects using a callback facility.
v. *Handling distributed I/O*: It is undesirable to have user code produce output in a remote machine or ask for input somewhere other than it is intended to. Therefore all input/output operations need to be redirected. This leads to the following possible transformations: Standard output and error, Standard input, File input and File output.
vi. *Exception handling*: Reference [14] illustrates the approach adapted for handling exceptions.
vii. *'final' class*: To extend the functionality of a un-modifiable class in the proxy, the final keyword is removed from the *classpool* [15] inside the class file. In this way, the semantic and functional consistency of the class remains the same but now extra methods can now be added and existing methods can be overridden in the proxy to add the extra functionality.
viii. *Existing interfaces*: Since Java allows a single class to implement as many interfaces as required, no changes are required in this case.
ix. *Static methods and fields*: Any class that has static methods and fields is divided into two subclasses where one has all the static methods and fields and another has the non-static entities. Separate proxies are created for each subclass and the static subclass is

anchored in one node and interacts with other proxies using RMI callbacks.

x. *Use of 'this' and 'super'*: Use of *super* does not cause any problem in the transformed code. However, the use of *this* needs to be delegated to the appropriate proxy class.

xi. *Use of reflection*: Reflection in the user code is not considered due to the added complexity in the byte code rewriting phase. A separate package on reflection that delegates user enforced reflection must be developed to address this issue.

To handle other Java language features, the techniques used have similarities with the techniques used in J-Orchestra [2]. The major distinction with the approach in this paper is that object level distribution is attempted, whereas J-Orchestra uses class-level distribution. One significant drawback with both of these approaches is that it incurs extra space and time cost to run the resulting distributed application. Since new proxy classes are created and segments of byte code are inserted into the exiting byte code, some inflation in the resulting code size is expected.

3.2 Distribution Transformation

Once a group of objects has been identified as a managed element of an autonomic element, the corresponding class files must be transformed accordingly to facilitate insertion of autonomic primitives into the code. The static portion of the class is first moved to a separate class by the static analyser [12] and corresponding redirection is established to preserve the semantic and functional flow of the original execution. Communication between distributed objects is permitted, but such interaction should be kept to a minimum in order to reduce the communication overheads. The assumption is that the distributed classes are multi-threaded to allow asynchronous execution. This assumption is true for the problem domain of the research where the whole program is modelled as a collection of concurrent objects.

For each of the distributed objects, the code transformer transforms the byte code by instrumenting code to provide autonomic properties. Specifically the points where distributed objects are declared and called are now transformed to call and invoke corresponding proxy object classes. The proxy can itself redirect that call to a distributed autonomic element depending on the system level distribution policy. Every distributable class is transformed and the distribution transformer makes the following changes:

- The distributed class is modified to implement a new interface which holds distribution primitives.
- Any input/output statements in the user code are marked for redirection as specified by system level policy. The reason for redirecting all input/output is explained in the next section.
- The constructor is replaced with an *initialize* method.
- Based on the parameter-passing mode (*pass-by-value* or *pass-by-reference*), all parameters and corresponding local variables are changed. Please see [11] for more details on the techniques used for parameter passing.
- All public methods in the distributed class are modified to throw a *RemoteException* (to satisfy RMI requirements) and *MigratedException* (to satisfy runtime migration) along with any other existing exception thrown by the method.
- To make remote objects migratable, they are transformed to implement the *Serializable* interface. The code transformer checks whether the class itself or any of the class in its inheritance tree implements *Serializable*, and adds it if necessary. However, not every class in Java can be made *Serializable* in that way because of the language level constraints. Such objects have to be anchored in one single node during the execution of the program.

Other than making these changes, the remainder of the code remains intact. As a requirement of RMI, all methods must be public so that they can be called remotely. So the code transformer checks that all access rights are maintained. Only after the semantic check and transformation, the methods in the generated class are made public.

4. Performance Analysis

Experiments were conducted to find the time requirement of delegated method calls using proxy-based managed elements. After code transformation, every remote method call now takes place through the autonomic element. Therefore the call can be broken down into several parts:

i. Call to local proxy.
ii. Call from local proxy to remote proxy (AE).
iii. AE locates the remote class, loads it and uses runtime reflection to delegate the execution to the corresponding method.

To explain the process, consider the UML diagram of an example program presented in Figure 1 (a). The application class (which is the starting point of this application) creates an instance of *classA* which is run as a threaded object. For instance, we like to transform the instance of *classA* into a managed element of an autonomic element. The class diagram is now transformed into three separate portions with separate transformations as follows:

1. *Application Class*: Instead of referring to *classA*, it will now refer to the proxy instance of *classA* (Figure 1 (b)). A new wrapper class is created for *classA* that acts as the proxy of the original class. Along with the traditional proxy transformation, a new private variable that holds the reference (local or remote) of the actual object, and a new method that helps initialize the actual object reference, is inserted into the proxy class.
2. *Used classes inside the managed instance*: Objects of any class that *classA* is using must now implement the *java.lang.Serializable* interface to help the system

to transfer the instance over the network if needed (Figure 1 (c). All such classes are checked to see whether they implement either the *Serializable* interface or a subclass of that interface. If not, then it is added automatically. If such classes use any object instance which is not serializable (for instance having another thread inside that object), then that particular managed element must be anchored on a single machine during the life time of the application.

3. *Original class*: The original class relationship is kept unchanged (Figure 1 (d)). Only the constructor is now replaced by a new method *initialize* to create an instance of the class during runtime by the autonomic element using a consistent interface.
   A new class is designed to house the managed object

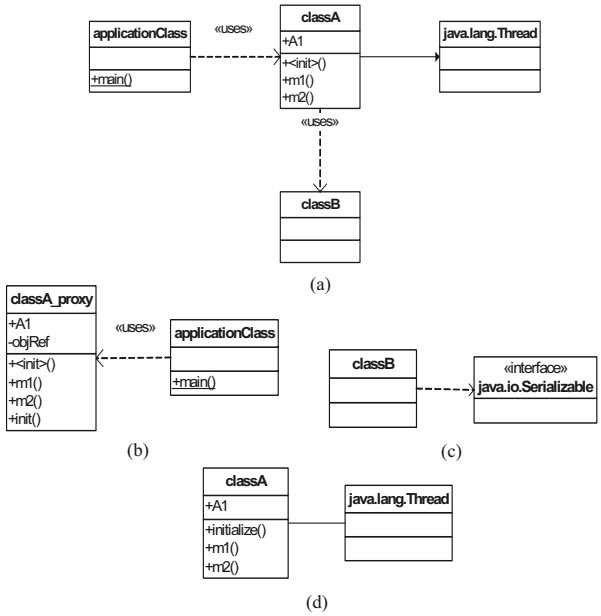


Figure 1. Example Class Diagram.

instance which implements a pre-defined interface. The *AutonomicElement* class creates an instance of the *managedElement* class. Figure 2 shows the runtime structure of each proxy class in the system. The *init* method in the proxy class is responsible for gathering necessary information (such as location of an available AE and naming of the AE) and establishing the deployment of the object to an autonomic element. Once all necessary information is gathered and processed, the proxy initially invokes the *setup* method of the corresponding *managedElement* with necessary information (such as class name for object instance, location of class and generated name for identifying that instance) as arguments. Next, the *init* method is invoked to create the actual object instance in the autonomic element with any initializing arguments. Any further method call from the proxy is then delegated by using the *invoke* method inside the proxy class. Figure 3 shows the timing for such remote calls, where other denotes the time it takes for local proxy class loading and locating the

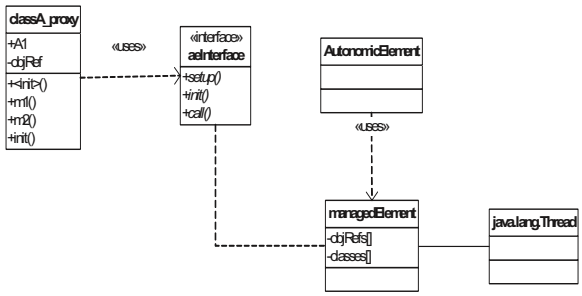


Figure 2. Transformed Runtime Program Structure.

remote AE using the standard naming protocol. For a single remote call using the delegated proxy pattern, the increase in execution time is approximately 45%. With several calls to the same AE, this reduces to less then 10% as the JVM normally caches classes and for subsequent calls class loading and reflection occur faster than the initial time. Table 1 shows this behavior of the proxy based remote invocation. Code inflation for the transformations is linier (around 10% for distribution transformation and 30% for autonomic transformation) and depends on the number of objects and classes in the user code.

The benefit of having a double delegated structure is that the distribution concerns and the self-management concerns are separated at the code level. This allows future code extension, since adding the two level delegations work in two separate processes and in sequence. Programmers can add their own transformations in the future if they wish following a similar approach; however the penalties for doing so are further increased code inflation and reduced response time due to extra code execution and class loading.

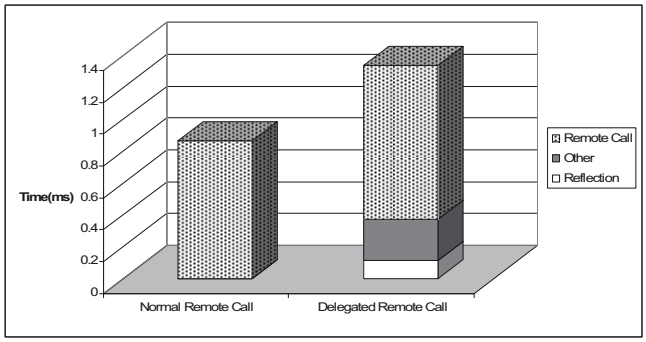


Figure 3. Timing Information for Delegated Invocation.

Table 1. Effects of Multiple Calls on Respnose Time.

Number of calls	Standard Remote Call (ms)	Delegated Remote Call (ms)	% Change
1000	391	476	21.74
10000	2289	2593	13.28
100000	19372	20261	4.59

5. Management Architecture

The design of the management architecture is driven by the desire to produce a transparent and easy to use

system that can be self-managed. The architecture provides services (physical resources) to the submitted workload (user application), where services are defined as an engagement of resources (service providers) for a period of time according to a contractual relationship (Service Level Agreement or SLA) with a service requester (application user). Figure 4 shows the overall system architecture. The architecture is broken into two views: *structural* and *managerial*. In the structural view, the lowest layer consists of the actual physical resources that are providing the services. This layer is abstracted by a virtual resource layer to hide the interface complexity of the actual physical devices. Having this virtual layer with a pre-defined interface allows the architecture to incorporate new physical devices in the future without re-programming the whole architecture for each new device. The upper layers provide a set of predefined services dependant on the state the current autonomic element is currently in. Some of the autonomic elements are directed to act as managerial elements to provide certain services across the entire system, so that other autonomic elements can work smoothly and can provide the service as requested. The application layer, normally kept dormant, can be brought alive if needed by either the user or by another autonomic element. The autonomic element layer is responsible for the management of autonomic elements and general system wide management. The middleware service layer performs actual communication services which are used for inter autonomic element communication, repository update, notification of migration of managed elements etc. The managerial view consists of the management functions necessary to

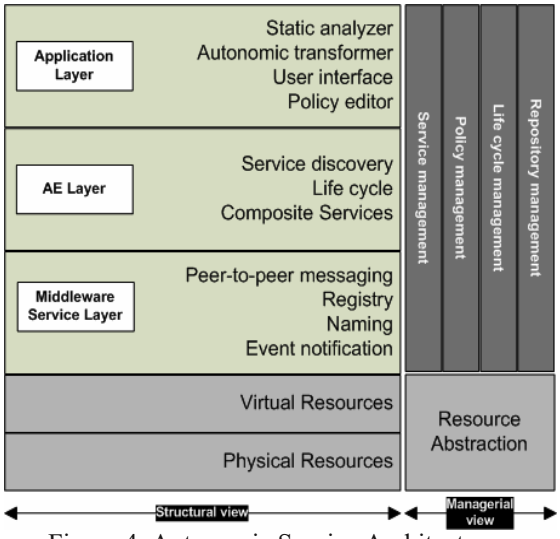


Figure 4. Autonomic Service Architecture.

manage the services delivered. Management functions primarily involve policy management, resource management, service management and life-cycle management. All management functions are performed by autonomic elements which are self-managing and whose role is to ensure automated delivery of services.

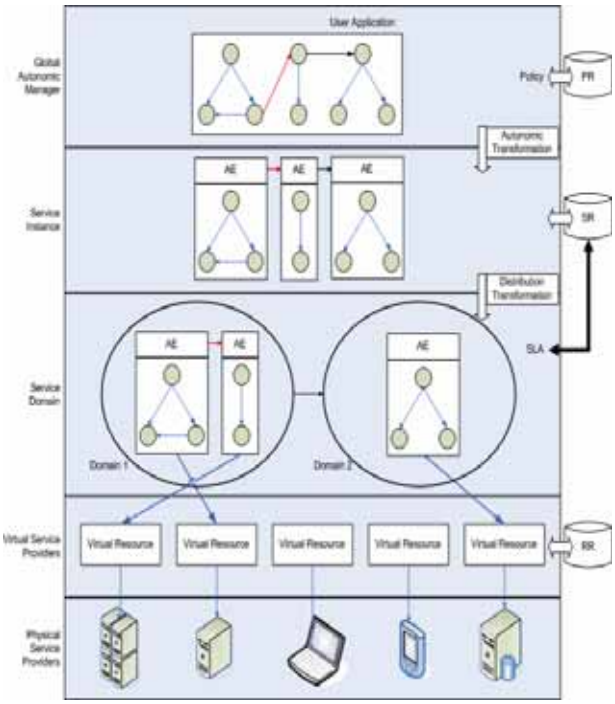


Figure 5. Hierarchical Management of System.

Figure 5 shows the hierarchical management view of the architecture. The global autonomic manager acts as the interface between the user and the underlying system. Each of the program partitions is treated as a service instance and encapsulated by an autonomic element for self-management. Each of the service instances are assigned to resource domains that best suit the instance's needs. See reference [13] for more information on devising such service instance to domain composition details. Each of the actual physical resources has a corresponding virtual resource adapter that provides a uniform and consistent interface to the physical resources. This simplifies resource management, service composition and dynamic resource addition/deletion. In order to perform autonomic service management, the system must maintain appropriate information about the different components of the runtime environment in repositories (databases). Repositories are classified into three distinct types:

- *Policy repository (PR)*: This repository contains the policies created at runtime or pre-defined by operators or users following a template policy implementation. The user can browse through existing policies and can use them to create new policies using a policy editor interface.
- *Service repository (SR)*: The service repository contains information (such as parties involved in a service level agreement) about the different service instances activated by the global autonomic manager, types of resource used, amount of resource being used and past operation history.
- *Resource repository (RR)*: This repository holds information about the resources available to the

service providers at any given time. Information that may be stored for each resource includes type of resource, resource performance matrices, communication matrices, etc.

The implementation of these repositories is performed hierarchically to avoid a single point of failure.

6. Management Operations

In this three tier peer to peer architecture (Figure 4), each autonomic element can also act as a managerial element. All autonomic elements operate over a unified management model that provides a set of operations and interface common to all autonomic elements. Normal autonomic elements access this model to retrieve their current configuration and save their current state, whereas managerial elements use it to discover other elements in the system. Representation of management information is a crucial part of any self-management architecture. The representation should be easy to manipulate and should follow an open standard for faster incorporation and future extensibility. XML-based WSDL is used to represent such management operations for these reasons. Currently, there are four types of management information kept by the system:

i. *Resource information*: Resource and service provider properties are expressed by this type of information. Care should be taken so that there are no duplicate entries and periodically (heuristically decided) this information needs to be updated to keep it consistent and non-redundant.
ii. *Performance information*: This type of information represents the performance status of a running autonomic element. Before two autonomic elements come to a service agreement, they transfer performance information to learn more about each other, which includes different performance measurements and current operational state.
iii. *Configuration information*: This type of information is used to configure the behavior of an autonomic element. Concurrency control has to be in place so that multiple configuration information is in play at the same time.
iv. *Relationship information*: This type of information expresses dependency relationships between autonomic elements.

A model similar to DNS revolver is employed to discover autonomic elements in the system.

7. Conclusions

The approach to transform code to add distribution concerns and self-management concerns is presented and evaluated in this paper. The software architecture to support such transformed code is also presented and different aspects of it is discussed. Architectural choices have a profound effect on the capabilities of any autonomic system and affect many of the design decisions during its implementation. The presented architecture supports computational and data intensive centralized applications where the computation-to-communication ratio is significant.

8. Reference

[1] Tatsubori, M., Sasaki T., Chiba S. and Itano L., "A Bytecode Translator for Distributed Execution of Legacy Java Software", *ECOOP 2001*, Hungary, pp. 236-255, 2001.

[2] Tilevich E. and Smaragdakis Y., "J-Orchestra: Automatic Java Application Partitioning", *ECOOP 2002*, Malaga, LNCS, Vol. 2374, pp.178-204, 2002.

[3]H. T. Feng and E. A. Lee, "Incremental Checkpointing with Application to Distributed Discrete Event Simulation", *Technical Report*, EECS Department, University of California, Berkeley, April, 2006.

[4] Julia L. L. and Gilles M., "Efficient Incremental Checkpointing of Java Programs", *Int. Conf. on Dependable Systems and Networks (DSN 2000)*, pp. 61-70, 2000.

[5] Peyman O., Nenad M. and Richard N. T., "Architecture-Based Runtime Software Evolution", *20th Int. Conf. on Software Engineering (ICSE'98)*, pp. 177 -186, 1998.

[6] M. E. Segal and O. Frieder, "Dynamic Program Updating: A Software Maintenance Technique for Minimizing Software Downtime", *Journal of Software Maintenance*, Vol. 1, No. 1, pp. 59-79, 1989.

[7] Tobias R. and Jesper A., "Dynamic Deployment of Java Applications", *Proocedings of Java for Embedded Systems*, London, May 2000.

[8] Kaffe VM for Java, http://www.kaffe.org/, 2006.

[9] Abbas N., Palankar M., Tambe S. and Cook J. E., "Infrastructure for Making Legacy Systems Self-managed", *Technical Report,* 2004.

[10] Griffith R. and Kaiser G., "Adding Self-healing Capabilities to the Common Language Runtime", *Computer Science Technical Report*, Columbia University, 2005.

[11] Fuad M. M.., "An Autonomic Software Architecture for Distributed Applications", *Ph. D. Dissertation*, Department of Computer Science, Montana State University, USA, 2007.

[12] Deb D., Fuad M. M. and Oudshoorn M. J., "Towards Autonomic Distribution of Existing Object Oriented Programs", *International Conference on Autonomic and Autonomous Systems*, IEEE Press, USA, pp. 17-23, 2006.

[13] Deb D., Fuad M. M. and Oudshoorn M. J., "ADE: Utility Driven Self-management in a Networked Environment", *Journal of Computers*, Academy Publishers, USA, Vol. 2, No. 8, 2007.

[14] Kephart J. O. and Chess D. M., "The vision of autonomic computing", *Computer*, Vol. 36, No. 1, pp.41–52, 2003.

[15] Fuad, M. M. and Oudshoorn, M. J., "Transformation of Existing Programs into Autonomic and Self-healing Entities", *The 14th IEEE International Conference on the Engineering of Computer Based Systems* (IEEE/ECBS), Arizona, USA, March 26 - 29, 2007, Pages 133-144.

[16] Sun Micro Systems, *Java Virtual Machine Specification*, 1999.

A Decision-Centric Architecture Design Method Facilitating the Contextually Capture and Reuse of Design Knowledge

Xiaofeng Cui, Yanchun Sun*, Sai Xiao, Hong Mei
Institute of Software, School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing 100871, China
{cuixf04, sunyc, xiaosai05}@sei.pku.edu.cn, meih@pku.edu.cn

Abstract

Software architecture design is a knowledge-intensive activity. Existing design methods mostly provide high-level processes and guidelines. The generic design knowledge, e.g., patterns, falls short of incorporating into specific design methods. Emerging research on Architectural Knowledge (AK) mostly focuses on its recording and comprehension, whereas how to leverage the knowledge to support architecting is still a very open issue. In this paper, we present a decision-centric architecture design method, explicate four kinds of design knowledge pertaining to this method, and propose a schema to record the knowledge. Because of their natural correspondence to the design notions and activities, these kinds of knowledge can be captured and retrieved for reuse at the right point of need, according to the design context. We illustrate how our architecture design tool automatically helps architects reserve the design knowledge and promotes potential reusable knowledge for them to accomplish the design more efficiently.

1. Introduction

Architecture design plays a crucial role in the whole lifecycle of software. Software development is a human and knowledge-intensive creative activity [1]. This claim is especially true for architecture design. Various kinds of knowledge, e.g., design knowledge, analysis knowledge, and realization knowledge, are all critical inputs to the design process [2]. The success and efficiency of the design depend on not only the tacit experience and skills of architects, but also the explicit form of the knowledge and the way to utilize it.

Existing architecture design methods [2] mostly provide high-level processes and guidelines for design, whereas still lacks pragmatic support for architects to discover solutions and make decisions. On the other hand, the wide-accepted design knowledge, e.g., patterns, though useful to codify successful expertise and improve architects' competence, is often too generic and lacks incorporation into specific design methods, so still falls short of supporting the architecting process directly and effectively.

The software architecture community has an emerging focus on Architectural Knowledge (AK), which is defined as the integrated representation of the software architecture of a software-intensive system or family along with architectural decisions and their rationale, external influence and the development environment [3]. Existing research on AK mostly focuses on the recording and comprehension of architecture decisions and rationale. How to leverage the accumulated knowledge to support architecting activities is still a very open issue.

We have proposed a decision-centric architecture design method [4], which models the core notions and provides a semi-automated process for architecture design. In this paper, we explicate four kinds of design knowledge specially pertaining to this design method, i.e., issue knowledge, solution knowledge, decision knowledge, and rationale knowledge. Because of their natural correspondence to the design notions and activities, these kinds of knowledge can be captured and retrieved for reuse at the right point of need, according to the design context. We have developed a tool to support the design method and leverage the design knowledge. We illustrate how this tool automatically helps architects reserve the design knowledge and promotes the potential reusable knowledge for them to accomplish the design more efficiently.

* Corresponding author

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 gives an overview of our decision-centric meta-model and architecture design process. Section 4 describes the design knowledge pertaining to our design method. Section 5 and 6 describes the capture and reuse of design knowledge within the design process. Finally, Section 7 presents concluding remarks and future work.

2. Related work

A number of methods have been proposed for software architecture design, e.g., QASAR [5], ADD[6], Siemens Four-Views (S4V) [7], RUP's 4+1 View [8], etc. Many methods derive the resulting architecture via a series of transformations and provide design knowledge in the forms of guidelines, patterns, tactics, etc. For example, QASAR [5] formulates design guidelines to indicate suitable transformations. ADD [6] applies architectural tactics and patterns that satisfy the driving quality attribute requirements. Our approach implements automated synthesis of candidate architecture solutions, and the capture and reuse of design knowledge during the design process.

The effort on Architectural Knowledge (AK) in the architecture community [3] is to make AK explicit to facilitate its sharing and reusing. Bosch [9] promotes that design decisions should be represented as first-class entities in software architectures. Kruchten et al. [10] describe a use-case model for an architectural knowledge base, together with its underlying ontology. Habli and Kelly [11] address the reuse of architectural knowledge through the use of derivational analogy. Ali Babar and Gorton [12] develop a framework and tool for capturing and using architectural knowledge to improve the architecture process. We tailor the architecture design knowledge to our specific design method, so that the knowledge can be leveraged efficiently for architecting.

Knowledge Management (KM) is an emerging discipline that promises to capitalize on organizations' intellectual capital, and the KM in software engineering is also drawn considerable attention [1]. Basili [13] proposes a framework where Experience Factory develops and packages experience for reuse. Henninger et al. [14] propose the organizational learning approach to software development. Rich and Waters [15] point out that the software engineering tools will need more knowledge intensive approaches. We follow the theme of knowledge-based approach for software development, and aim to take advantage of the knowledge in software architecture design with a lightweight knowledge management effort.

3. Overview of the decision-centric architecture design method

Figure 1 shows the meta-model to describe the notions in our decision-centric architecture design.

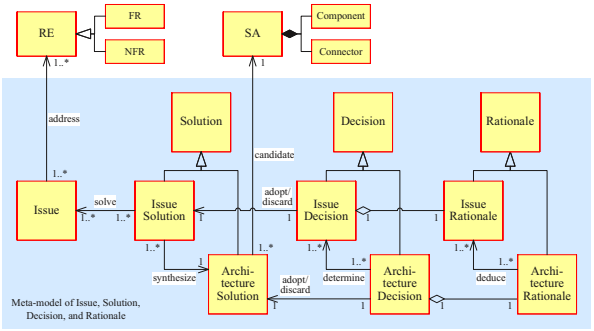


Figure 1. The decision-centric meta-model

An *issue* is an architecturally significant problem that must be solved by the architecture design. Issues address specific aspects of requirements or any other considerations at the architecture level.

Solutions are specialized into *issue solutions* and *architecture solutions*. An issue solution provides a possible way of solving an issue. An architecture solution is a candidate architecture design that has addressed all of the issues. Architecture solutions can be synthesized from issue solutions.

Decisions are specialized into *issue decisions* and *architecture decisions*. An issue decision means adopting or discarding one candidate issue solution. An architecture decision means adopting or discarding one candidate architecture solution. Issue decisions can be determined by architecture decisions.

Rationale is specialized into *issue rationale* and *architecture rationale*. Issue rationale is the reason behind issue decisions. Architecture rationale is the reason behind architecture decisions. Issue rationale can be deduced from architecture rationale.

Figure 2 shows the iterative process to implement our decision-centric architecture design. The main inputs to this process are software requirements and (optional) original architecture. The main outputs of this process are decided architecture, recorded decisions, and captured rationale.

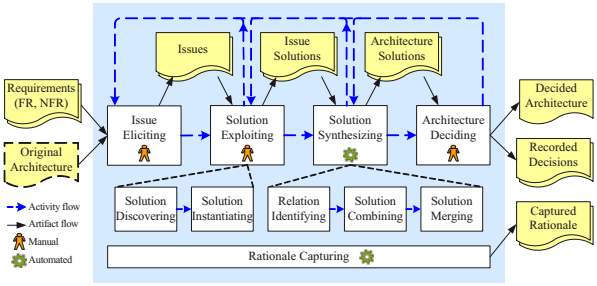


Figure 2. The decision-centric design process

In the *issue eliciting* activity, stakeholders deliberate and determine architecturally significant issues, based on the requirements, the original architecture, and the design considerations.

In the *solution exploiting* activity, architects derive candidate solutions to each issue. First, they discover solutions, according to reusable design knowledge or newly developed technologies. Second, they instantiate these solutions from informal descriptions to concrete design models.

The *solution synthesizing* activity automatically synthesizes the candidate architecture solutions from various issue solutions. First, the relations between issue solutions (e.g., inclusive, conflictive, etc.) are identified to indicate whether they can be combined together. Second, a combination tree is built to explore all feasible combinations of issue solutions. Finally, the issue solutions within every feasible combination are merged to generate candidate architecture solutions.

In the *architecture deciding* activity, stakeholders select the target architecture solution from the synthesized candidate architectures. They need to evaluate the candidate architecture solutions according to the requirements, compare them by multiple criteria, make trade-offs between the competing objectives, etc.

The *rationale capturing* activity automatically deduces issue decisions and rationale from the settled architecture decisions and rationale, based on the synthesis relationship between architecture solutions and issue solutions. If one issue solution participates in synthesizing the architecture solution that is adopted, then the decision on this issue solution is adopting too. Otherwise the decision on this issue solution is discarding. Moreover, the rationale of the architecture decisions can be used to explain the reasons for adopting or discarding the issue solutions that participate in synthesizing the architecture solutions.

4. The design knowledge pertaining to the design method

The design knowledge here refers to the problem-solving expertise within our architecture design process. We explicate four kinds of design knowledge: *issue knowledge* (illuminates how to elicit issues from requirements), *solution knowledge* (illuminates how to exploit issue solutions and architecture solutions), *decision knowledge* (illuminates how to make architecture decisions and issue decisions), and *rationale knowledge* (illuminates the rationale of architecture decisions and issue decisions).

To preserve all these kinds of design knowledge, we codify comprehensive information about each architecture design, including not only the resulting architecture, but also the decisions and rationale, the candidate solutions, the issues, and the requirements, etc. Furthermore, we record all the relations between these entities, as specified in the meta-model, including the tracing relations from issue solutions to issues and to requirements, the synthesizing relations from issue solutions to architecture solutions, the deducing relations from architecture decisions and rationale to issue decisions and rationale, etc.

We define the structure of above information with an XML schema, where the architecture design repository (*ADRepository*) includes information about every architecture design project (*ADProject*). Figure 3 shows the schema of *ADProject*.

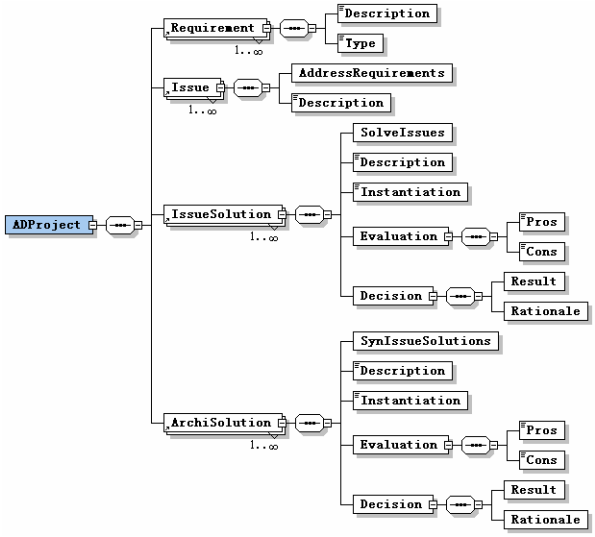


Figure 3. The XML schema of design information

The *ADProject* schema specifies four kinds of information items:

- *Requirement*, represents one requirement. The *Type* specifies whether it is a functional (*FR*) or non-functional requirement (*NFR*).
- *Issue*, represents one issue. The *AddressRequirements* specifies the requirements it addresses.
- *IssueSolution*, represents one candidate issue solution. The *SolveIssues* specifies the issues it solves. The *Instantiation* is an URI pointing to the concrete ADL model of this solution. The *Evaluation* includes *Pros* and *Cons* of this solution. As aforementioned, the issue decision is determined by the decisions on the architectures that this issue solution participates in synthesizing. If this issue solution participates in synthesizing the architecture solution that is adopted, the *Result* of this issue solution's *Decision* is *ADOPT*, otherwise is *DISCARD*. The *Rationale* is the link to the *Decision*s of the architecture solutions that this issue solution participates in synthesizing.

- *ArchiSolution*, represents one candidate architecture solution. The *SynIssueSolutions* specifies the issue solutions that participate in synthesizing this architecture solution. The *Instantiation* is an URI pointing to the concrete ADL model of this solution. The *Evaluation* includes the *Pros* and *Cons* of this solution. The *Result* of this architecture solution's *Decision* may be *ADOPT* or *DISCARD*. This *Result* is specified by the architects. The *Rationale* is a statement of the reason behind this decision, and it is also specified by the architects.

The above comprehensive information manifests the four kinds of design knowledge. In concrete terms, the knowledge is embodied in these information items and their correlations: the issue knowledge is embodied in the *Issue* items, and their relations to the requirements they address (*AddressRequirements*); the solution knowledge is embodied in the *IssueSolution* items and their relations to the issues they solve (*SolveIssues*), as well as the *ArchiSolution* items; the decision knowledge and rationale knowledge is embodied in the *Decision* and *Rationale* elements of the *IssueSolution* and *ArchiSolution* items.

Section 5 and 6 illustrate these kinds of knowledge, as well as their capture and reuse, based on a case of Commanding Display System (CDS) [4]. The target of CDS architecture design is to achieve a real-time, high dependency system for live and history data display.

5. Capture of the design knowledge within the design process

Capture issue knowledge. The issue knowledge can be captured within the context of issue eliciting activity. Our tool provides interfaces for users to specify the elicited issues, as shown in Figure 4. The tool then automatically records the issues and their relations to the requirements. Figure 5 shows a sample of the recorded requirements and issues that embody issue knowledge.

Capture solution knowledge. The solution knowledge can be captured within the context of solution exploiting and solution synthesizing activities. In the solution exploiting activity, our tool provides interfaces for users to specify the exploited issue solutions. The tool then automatically records the solutions and their relations to the issues. In the solution synthesizing activity, the tool automatically synthesizes candidate architecture solutions from various issue solutions, and records the synthesized architecture solutions. Figure 6 shows a sample of the recorded issue solutions and architecture solutions that embody solution knowledge.

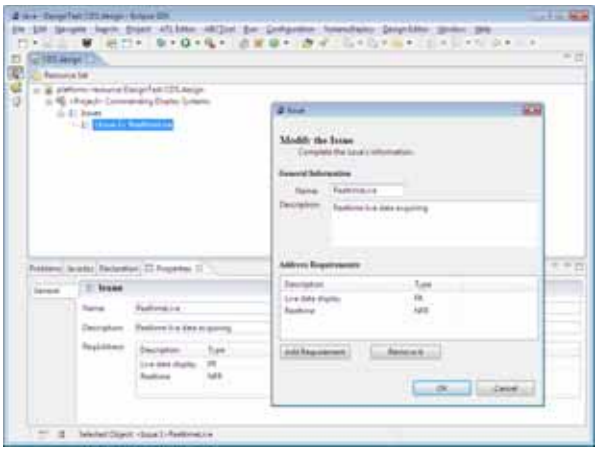


Figure 4. Interfaces for specifying issues

```
<Requirement ID="RE0001">
   <Description>Live data display</Description>
   <Type>FR</Type>
</Requirement>
<Requirement ID="RE0004">
   <Description>Real-time</Description>
   <Type>NFR</Type>
</Requirement>
<Issue ID="IU0001">
   <AddressRequirements IDREFS="RE0001 RE0004"/>
   <Description>Real-time live data acquiring</Description>
</Issue>
```

Figure 5. The captured issue knowledge

```
<IssueSolution ID="IS0001">
   <SolveIssues IDREFS="IU0001"/>
   <Description>DS pushes data to Ms directly</Description>
   <Instantiation>IS0001.ADL</Instantiation>
   <Evaluation>
      <Pros>Simple to implement; normal DB technology</Pros>
      <Cons>Lack of management for live data</Cons>
   </Evaluation>
   <Decision>
      <Result>DISCARD</Result>
      <Rationale xlink:type="simple" xlink:href="xpointer(//
       ArchiSolution[contains(SynIssueSolutions/@IDREFS,
       here()/../@ID)]/Decision)"/>
   </Decision>
</IssueSolution>
<ArchiSolution ID="AS0008">
   <SynIssueSolutions IDREFS="IS0002 IS0005 IS0007
    IS0009 IS0011"/>
   <Description>A real-time DB is used to store all data from
    DS, Ms get live data from DB directly, …</Description>
   <Instantiation>AS0008.ADL</Instantiation>
   <Evaluation>
      <Pros>Advantage of real-time DB, maintainable, …</Pros>
      <Cons>Cost of development, …</Cons>
   </Evaluation>
   <Decision>
      <Result>ADOPT</Result>
      <Rationale>New technology, high maintainability.
      </Rationale>
   </Decision>
</ArchiSolution>
```

Figure 6. The captured solution knowledge, and the decision and rationale knowledge

Capture decision and rationale knowledge. The decision and rationale knowledge can be captured within the context of architecture deciding and rationale capturing activities. In the architecture deciding activity, our tool provides interfaces for users to specify their decisions on the candidate architecture solutions and the rationale of their decisions. The tool then automatically records these decisions and rationale. Moreover, the tool automatically deduces issue decisions and rationale from the architecture decisions and rationale, and records them. Figure 6 also shows a sample of the recorded decisions and rationale.

6. Reuse of the design knowledge within the design process

Reuse issue knowledge. The issue knowledge can be reused in the issue eliciting activity, to help users elicit issues according to the specified requirements. Within this context, our tool automatically retrieves potential reusable issues from the design repository via a background query. The query finds the issues whose *AddressRequirements*s include the *Requirement*s whose *Description*s include the keywords of the current project's requirements.

Besides the automatically suggested issues, users can also conduct several kinds of queries manually to find other potential reusable issues. For example:

- Query the *Issue*s whose *Description*s include the specified keywords.
- Query the *Issue*s whose *AddressRequirements*s include the *Requirement*s whose *Description*s include the specified keywords.

Figure 7 shows the interface automatically suggesting issues for reuse when users specify a new issue (the left dialog), and the interface for users to query the issue knowledge manually (the right dialog).

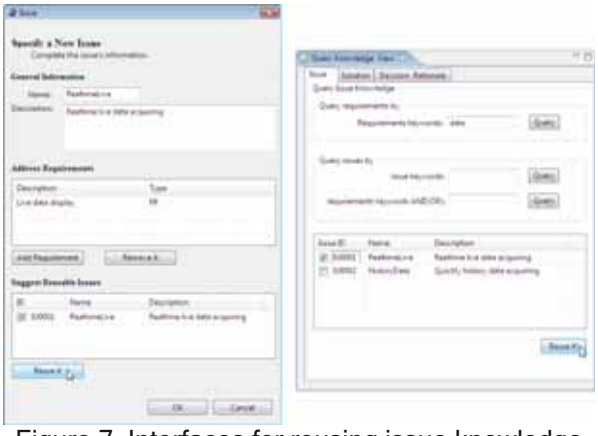


Figure 7. Interfaces for reusing issue knowledge

All above queries can be implemented on the recorded issue knowledge. For example, a query of the *Issue*s by the requirement keyword "Live" is implemented with XPath:

```
//Issue[contains(AddressRequirements/@IDREFS,//Requirement[contains(Description,"Live")]/@ID)][//Requirement[contains(Description,"Live")]]
```

Reuse solution knowledge. The solution knowledge can be reused in the solution exploiting activity, to help users exploit solutions to the specified issues. Within this context, our tool automatically retrieves potential reusable solutions from the design repository. For the reused issues, the tool retrieves their solutions in the design repository directly and promotes for reuse. For the newly elicited issues, the tool query the issues whose *Description*s include the keywords of the new issues, then retrieves the solutions to these similar issues and promotes for reuse.

Besides the automatically suggested solutions, users can also conduct several kinds of queries manually to find other potential reusable issue solutions. For example:

- Query the *IssueSolution*s whose *Description*s include the specified keywords.
- Query the *IssueSolution*s whose *SolveIssues*s include the *Issue*s whose *Description*s include the specified keywords.
- Query the *IssueSolution*s that are included in the *SynIssueSolutions*s of the *ArchiSolution*s whose *Description*s include the specified keywords.

Architecture solutions are not prone to be directly reusable, because of their relative coarse granularity and the diversity of different projects. However, they can also be queried to provide references. For example:

- Query the *ArchiSolution*s whose *Decsription*s include the specified keywords.
- Query the *ArchiSolution*s whose *SynIssueSolutions*s include the *IssueSolution*s whose Descriptions include the specified keywords.

All above queries can be implemented on the recorded solution knowledge. For example, a query of the *ArchiSolution*s by the issue solution keyword "push" is implemented with XPath:

```
//ArchiSolution[contains(SynIssueSolutions/@IDREFS,//IssueSolution[contains(Description,"push")]/@ID)][//IssueSolution[contains(Description,"push")]]
```

Reuse decision and rationale knowledge. The decision and rationale knowledge can be reused in the architecture deciding activity, to help users make decisions on the specified solutions. Within this context, our tool automatically retrieves potential reusable decisions and rationale from the design repository. For the reused issue solutions, the tool

retrieves their decisions and rationale directly and provides references for users.

Besides the automatically suggested decisions and rationale, the user can also conduct several kinds of queries manually to find the relative decisions and rationale for reference. For example:

- Query the *Decision*s and *Rationale* on the *IssueSolution*s by the decision *Result*.
- Query the *Decision*s and Rationale on the *IssueSolution*s whose *Description*s include the specified keywords.

As aforementioned, architecture solutions are not prone to be directly reusable, neither are their decisions and rationale. However they can also be queried to provide references for users. For example:

- Query the *Decision*s and *Rationale* on the *ArchiSolution*s by the decision *Result*.
- Query the *Decision*s and *Rationale* on the *ArchiSolution*s whose *Description*s include the specified keywords.

All above queries can be implemented on the recorded decision and rationale knowledge. For example, a query of the *Decision*s on the architecture solutions by the keyword "real-time DB" is implemented with XPath:

```
//ArchiSolution[contains(Description,"real-time DB")]/Decision
```

7. Conclusions and future work

We have proposed a decision-centric architecture design method and explicated four kinds of design knowledge pertaining to this design method. The architecture design method can facilitate the capture and reuse of the design knowledge within the context of various design activities. The tool we developed can provide semi-automated support for the architecting and the capture and reuse of the design knowledge.

This paper presents our ongoing work towards practical architecture design method and tool support. Future work includes the further elaboration of the architecture design knowledge and the validation of this method and tool in real-life applications.

8. Acknowledgements

This effort is sponsored by the National Basic Research Program (973) of China under Grant No. 2005CB321805, the National Natural Science Foundation of China under Grant No. 90612011, 60503028, and the National High-Tech Research and Development Program (863) of China under Grant No. 2007AA01Z127, 2007AA010301.

9. References

[1] I. Rus and M. Lindvall, "Knowledge Management in Software Engineering," *IEEE Software*, vol. 19, no. 3, May/June 2002, pp. 26-38.

[2] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, "A General Model of Software Architecture Design Derived from Five Industrial Approaches," *The Journal of Systems and Software*, vol. 80, no. 1, Jan. 2007, pp. 106-126.

[3] P. Avgeriou, P. Kruchten, P. Lago, P. Grisham, and D. Perry, "Architectural Knowledge and Rationale – Issues, Trends, Challenges," *ACM SIGSOFT Software Engineering Notes*, vol. 32, no. 4, July 2007, pp. 41-45.

[4] X. Cui, Y. Sun, and H. Mei, "Towards Automated Solution Synthesis and Rationale Capture in Decision-Centric Architecture Design," *Proc. Working IEEE/IFIP Conf. on Software Architecture (WICSA'08)*, IEEE CS, Feb. 2008, pp. 221-230.

[5] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.

[6] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley, 2003.

[7] C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*. Addison-Wesley, 2000.

[8] P. Kruchten, *The Rational Unified Process: An Introduction*, 3rd ed. Addison-Wesley, 2003.

[9] J. Bosch, "Software Architecture: the Next Step," *Proc. 1st European Workshop on Software Architecture (EWSA'04)*, Springer-Verlag, 2004, pp. 194-199.

[10] P. Kruchten, P. Lago, and H. v. Vliet, "Building Up and Reasoning About Architectural Knowledge," *Proc. 2nd Int'l Conf. on the Quality of Software Architectures (QoSA'06), LNCS 4214*, Springer-Verlag, 2006, pp. 43-58.

[11] I. Habli and T. Kelly, "Capturing and Replaying Architectureal Knowledge through Derivational Analogy," *2nd Workshop on SHAring and Reusing architectural Knowledge - Architecture rationale and Design Intent (SHARK-ADI'07)*, IEEE CS, May 2007.

[12] M. A. Babar and I. Gorton, "A Tool for Managing Software Architecture Knowledge," *2nd Workshop on SHAring and Reusing architectural Knowledge - Architecture rationale and Design Intent (SHARK-ADI'07)*, IEEE CS, May 2007.

[13] V. R. Basili, "Software Development: A Paradigm for the Future," *Proc. 13th Int'l Computer Software and Applications Conf. (COMPSAC'89)*, IEEE CS, 1989, pp. 471-485.

[14] S. Henninger, K. Luppala, and A. Raghavendran, "An Organizational Learning Approach to Domain Analysis," *Proc. 17th Int'l Conf. on Software Engineering (ICSE'95)*, ACM, 1995, pp. 95-104.

[15] C. Rich and R. C. Waters, "Knowledge Intensive Software Engineering Tools," *IEEE Trans. on Knowledge and Data Engineering*, vol. 4, no. 5, Oct. 1992, pp. 424-430.

System architecture induces document architecture

Peter Henderson, Nishadi De Silva
School of Electronics and Computer Science, University of Southampton
Southampton, UK
p.henderson@ecs.soton.ac.uk, n.desilva@ecs.soton.ac.uk

Abstract

The documentation of an architecture is as important as the architecture itself. Tasked with communicating the structure and behaviour of a system and its constituent components to various stakeholders, the documentation is not trivial to produce. It becomes even harder in open, modular systems where components can be replaced and reused in each progressive build. How should documentation for such systems be produced and how can it be made to easily evolve along with the system it describes? We propose that there is a close mapping between the system architecture and its documentation. We describe a relational model for the architecture of open systems, paying close attention to the property that certain components can be reused or replaced. We then use ideas from storytelling and a discourse theory called Rhetorical Structure Theory (RST) to propose a narrative-based approach to architecture documentation; giving both a generic narrative template for component descriptions and a RST-based relational model for the document architecture. We show how the two models (system and documentation) map onto each other and use this mapping to demonstrate how document fragments can be stored, automatically extracted and collated to closely reflect the system's architecture.

Keywords

System architecture, documentation, narratives, RST

1. INTRODUCTION

An architecture is the partitioning of a whole into parts (components), with specific relationships between these parts [1-3]. There is an increasing need for faster software development, and much of this is now dependent on modular architectures with reusable components that allow for quicker evolution and localised updates [4]. Documenting the architectures of such evolving systems is not trivial. Of all the potential stakeholders, we are concerned primarily with the documentation required by developers who are charged with evolving the product. So, the question we ask is - how does one produce documentation for a developer who has to revise the software and thus use most of its documentation?

There are various techniques and guidelines on how to document architectures [1, 5-7]. Our approach, however, looks at this problem from a narratives perspective based on the hypothesis that 'saying it like a story' improves document coherence and readability. There are two issues that need to be considered: each component needs to be documented well and coherently; and, secondly, these component descriptions need to be collated in some way to produce the documentation for a system. For the first, we argue that a document conveys an implicit narrative (or story) to the reader, and that fine-tuning this improves the overall document. We use ideas from Rhetorical Structure Theory (RST) [8] to study and enhance the coherence of this implicit narrative (which we call a **document narrative or DN**) [9]. In this paper, we present a generic DN to document a component's structure and behaviour.

To address the second issue, we develop a relational model for the system architecture (comparable to other relational models in this field [10]) and a RST-based relational model for the document architecture, and show how the two map onto each other. We use this mapping to describe how aspects of the system architecture can be used to guide the structure and sequence of the documentation.

A mapping between the two models as shown here has two major benefits. Firstly, it allows a database to be created that can store the architecture details and the set of associated document fragments. When queried, it is able to return a narrative-based document that reflects the system architecture. Better still, it allows documentation to be reused or replaced where appropriate. Secondly, since there is a strong correlation between the models, system architects will be forced to think of the accompanying documentation from an early stage which will benefit both the system and the documentation. We conjecture that architectures that are easier to document using our technique are better architectures.

The rest of this paper is set out as follows: Section 2 gives some background information; section 3 introduces the relational model for the system architecture and a generic DN for documenting a component; section 4 presents the RST-based relational model for the documentation and illustrates the mapping between the two models; in section 5, we demonstrate our ideas using a simple example and section 6 concludes the paper and discusses future work.

2. BACKGROUND

A significant proportion of a software architect's time is spent interacting with stakeholders and communicating the architecture [11]. A majority of this communication is done via documentation. Architecture documentation is expected to cater to three categories of readers: those selecting this system, those learning to develop typical applications using this system and those intending to modify its architecture

[6]. The work presented in this paper addresses documentation targeted at the third category (even though it could be of use to the first group too).

Because architectures can be so complex, several practitioners and researchers have developed techniques that divide the documentation into **views** which help separate the different aspects of the architecture [1, 5, 12]. The documentation is then composed of the relevant views along with any documentation that applies to more than one view (the 'glue' that binds the views together). Similarly, Kruchten introduced a 4+1 model for an architecture [5] which is a generic way to describe architecture using five concurrent views, each addressing a specific set of concerns important to different stakeholders: the logical view, process view, physical view, development view and a fifth view that contains use cases or scenarios.

We recognise from these previous approaches that it is impossible to capture everything about an architecture in one document. We, therefore, abstract away from development and physical details to a much higher level. At this level, we only focus on descriptions about the software components, what they are made up of and how they interact with other components. We recognise that other audiences may require other types of documentation but they are beyond the scope of this paper.

We are also not the first to employ documenting strategies from another domain in architecture documentation. The **pyramid principle** [13], for instance, has been used to structure architecture documentation [6]. The pyramid principle is based on structuring the document around developing a question-answer dialogue with the reader. So, information is exposed incrementally as answers to questions that arise in the reader's mind. Also, **storyboarding** has been used to identify requirements and select COTS components [14]. In this paper, we make use of our previous work on **narrative-based writing** [9] and apply it to architecture documentation. This combination of narratives and RST in this domain is a novel approach. (A brief introduction to RST is given in section 4 and the features of narrative-based writing required for this paper are included where necessary. More can be found in [9].)

3. A RELATIONAL MODEL FOR SOFTWARE

As with most architectural descriptions, the central concept in our model is a **component**. A component can either be atomic or have subcomponents plugged into appropriate **slots**[1]. These subcomponents, in turn, can be made of sub-subcomponents and so on. This continues until a level is reached where the components can be considered as 'black boxes' (i.e., it is unnecessary and beyond the scope of the documentation to dwell deeper into the hierarchy of decomposition). This leads us to the first relation in our model:

contains (container:component, slot, component:component)

Components also have dependencies on other components. This is, in fact, essential for modular systems where the behaviour of the whole is only realised when the constituent components work together. We call this the *uses* relation. Component A *uses* B if A (user) uses an interface provided by B (service).

uses (user:component, service:component)

A particular benefit with open, modular architectures like the ones we focus on is that a component can be replaced by another component if it provides the similar functionality and interfaces. This can happen, for example, when two suppliers manufacture comparable components leaving the implementer to pick one depending on other criteria such as price and reliability. Of course, this option to replace usually works only in one direction. A superior component B' that can perform all the functions of an inferior component B (and more) can be used to replace B. However, B cannot be used in situations where a B' is required. This brings about the third relation *replaces*:

replaces (superior:component, inferior:component)

A diagrammatic representation of the three relations is shown below.

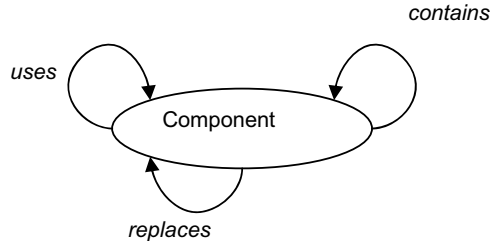


Fig. 1. Diagrammatic representation of our relational model for the system architecture

We realise that, when compared to languages such as UML with numerous relations, our model may appear limited. However, for the purposes of this paper, the model given above is sufficient.

4. A RELATIONAL MODEL FOR DOCUMENTATION

In our previous work, we have researched and developed a technique called narrative-based writing [9] to improve the coherence of technical documents such as research proposals. The technique required authors to first formulate a "document narrative" (DN): an explicit précis of what the authors wanted to convey to the readers in a story-like

[1] The idea of a 'slot' gives us a the flexibility to have multiple subcomponents of the same type plugged into different slots within the same component.

form. The DN is then analysed using a discourse theory called Rhetorical Structure Theory (RST) [8]. RST helps add more meaning and supportive reasoning to the DN and also gives an indication of how well it is structured. The DN and the corresponding RST analysis are then used to produce the document. The technique was particularly useful in collaborative writing where multiple authors had differing opinions about the document's objectives and structure.

We use this technique here to compose fragments of documentation corresponding to the components in the architecture. However, before proceedings, it is necessary to give a brief overview of RST and how it can be applied to text.

Rhetorical Structure Theory (RST)

RST was developed in 1988 by Mann and Thompson [8]. The theory attributes the coherence of a text to implicit logical relationships that exist between parts (usually called segments) of that text. So, for instance, segment A and B can be involved in a MOTIVATION relationship which means that segment B provides some information to motivate the action(s) in segment A. In Mann and Thompson's original paper, they define 23 such relationships with precise definitions of the sorts of text that can be involved in each.

In RST, the segments of text are classified as nuclei or satellites. Nuclei are considered essential to the understanding of the text. Satellites provide supporting material to the nuclei but are not absolutely necessary. Most relationships exist between a nucleus and a satellite. Returning to the example of the MOTIVATION relationship before, it can be illustrated using the diagram below. Note that the arrow always goes from the satellite to the nucleus.

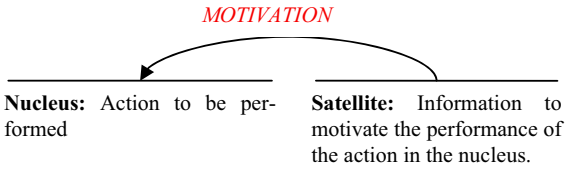


Fig. 2. A MOTIVATION relationship in RST

Some relationships like SEQUENCE can exist between more than two segments of equal importance (so, two or more nuclei). We have briefly described the RST relationships that appear in this paper in Table 1.

In order to do a RST analysis, the first step is to divide the text into segments. Each segment should have functional integrity and is often a clause or a sentence. The next step in a bottom-up analysis is to identify relationships that exist between pairs of segments. Segments involved in a relationship can, in turn, become involved in another relationship. Hence, the process is recursive and continues until all the segments can be assembled into a tree of relationships called a RS-tree. Mann and Thompson conjecture that if a RS-tree can be formed involving all the segments, then the text is coherent. However, if there are non-sequitors or difficulties producing this tree, then the text may need restructuring. This is a valuable guide when evaluating the structure and coherence of a text [8].

Relationship	Description
Background	Satellite provides background information to the nucleus
Elaboration	Satellite elaborates the information in the nucleus
Justify	Satellite justifies the information presented in the nucleus
Motivation	Satellite motivates the reader to perform the action in the nucleus
Sequence	Multiple nuclei that follow each other in sequence
Restatement	Satellite is a restatement of the information in the nucleus

Table 1. The RST relationships used in this paper

A Narrative-based Component Description

We look first at applying the narrative-based writing technique to describing each component. What we want to end up with is a generic structure that can be used for all components. Bearing in mind that a 'component' in our case can mean anything from a composite system to an atomic sub-component, some of the key concepts that need to be conveyed in the documentation are its behaviour, subcomponents (if any), whether it is able to interact with other components and, if appropriate, brief comparisons to similar products that are available. However, what is the best order to place this information in? This is where a DN can help. Trying to construct a narrative helps identify the natural sequence to the information and even recognise segments that are missing. A generic DN for the component descriptions (divided into 7 segments) is presented below along with a possible RST analysis of it. We say "a possible analysis" because it is viable that different analysts will produce different RS-Trees. The important point is to agree with the co-authors on the analysis and be able to form a tree (see Figure 3) which helps gauge the level of coherence of the text.

"[Select component X][1] [because it meets the set requirements and has some advantages over comparable technologies in the market.][2] [It is also a vast improvement from previous versions.][3] [It can receive the following instructions and perform the necessary tasks in response.][4] [The behaviour was grouped as it is done in this component for several good reasons.][5] [Furthermore, X can also interact with other components that it needs to in the following ways to produce the desired effect.][6] [On closer inspection, X is composed of multiple subcomponents that, when combined, enable its functionality. These components are x_1-x_n and they will be described later.][7]"

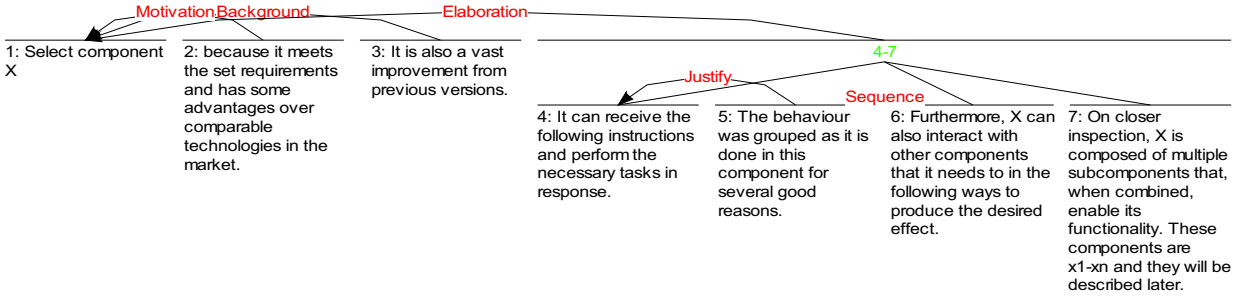


Fig. 3. A possible RST analysis of the generic DN above

Some parts of the narrative may not apply to all components of course. For instance, when describing components that are not going to be further decomposed, segment 7 about subcomponents is not relevant. Segment 2 is seen to provide motivation to convince the reader to choose (or buy) component X in the case where a decision has not yet been made.

It is worth mentioning that this narrative structure applies to the *body* of the document. Additionally, there would be other sections such as the introduction and conclusions which are compulsory in most documents. We call the description of a component adopting this narrative a **FRAGMENT**. A fragment is a self-contained description of an architectural component. Note that a fragment will be divided into several segments prior to doing a RST analysis. For a structured component, the fragments describing its contained components will be organised into a narrative structure where the fragments at the lower level are taken to be RST segments at the higher level.

A Relational Model for Document Architectures

From the above, we see that, for an architecture involving many different components at different levels in the hierarchy, there will be as many document fragments. For a document about the architecture, several of these fragments will need to be placed in a suitable order. Our eventual target is to develop a system where document fragments can be automatically extracted according to the architecture. To this end, we have developed a relational model for the documentation that corresponds to the system architecture. The novelty about this model is that these relations are also from RST. A fragment is central to our documentation model. Conceptually, this is similar to the component in the system architecture model.

Firstly, it needs to be noted that a fragment can be made up of other fragments. This is similar to the *contains* relation in the system architecture except that in the document model, a fragment's *narrative* is composed of other fragments' *narratives*. So, the topmost fragment will contain an description of the system and this is elaborated by fragments about overview of the system which is expanded by subsequent fragments (like sub-sections). We equate this to the RST ELABORATION relationship.

elaboration (fragment, fragment)

For components at the same level, the corresponding fragments need to be presented in an appropriate sequence. We propose using the *uses* relationship from the system architecture to determine the sequence. So, if component A uses component B, then we propose that the most suitable way to document it is to make fragment(A) appear before fragment(B). We call this second relationship SEQUENCE (also a RST relationship). We need to break loops in the *uses* relation by a suitable forward-reference mechanism. We recognise that even then the *uses* relation is only a partial order, but it seems not to matter which order unrelated fragments appear, as long as all the descriptions of the components that use them appear first.

sequence (fragment, fragment)

If components can be replaced by other components, it must be the case that the corresponding fragments can be replaced too. However, it is important to note that the replacement of document fragments works in the *opposite direction* to the replaces in the system architecture. Say, for instance, a newer component A' with more functionality is used to replace component A in a build. However, if fragment(A') is not yet ready, it is still possible to use fragment(A) in this case because only the capabilities of A are expected and realised. However, fragment(A) cannot be used in an instance where A' is required because it will not describe the extended functionality. The closest relationship in RST for this is RESTATEMENT. In RST, this means that one segment says the same thing as another in a different way.

restatement (fragment, fragment)

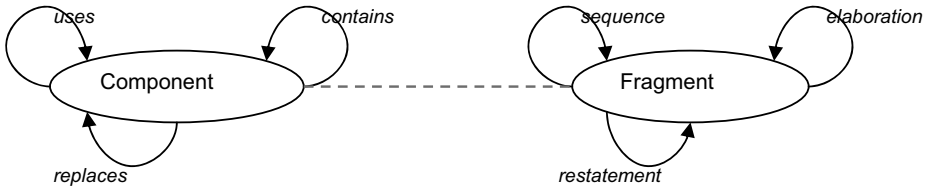


Fig. 4. The mapping between the system architecture model (left) and the document architecture model (right)

The figure above shows the mapping between the system architecture model and the document architecture model.

5. A SIMPLE EXAMPLE

We demonstrate the storage and extraction of document fragments using a simple example of a toaster T. T is made up of two subcomponents: the heating element (H) and the control module (C) which instructs H to start heating when the lever is pushed (thus, C uses H). Furthermore, H has a sub-subcomponent M, the timer.

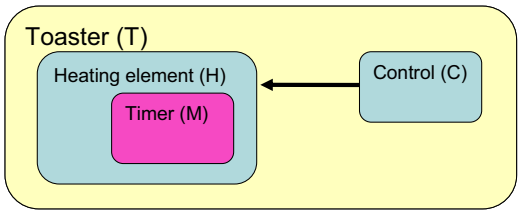


Fig. 5. A simple toaster T

Additionally, we know that a newer version of C, C', that can respond to changes in the 'browning level' made by the user can replace C. Similarly H' is more advanced and can vary the time of heat depending on the browning level. This information can be recorded using relational tables:

contains

container	slot	component
T	h	H
T	c	C
H	m	M

uses

user	service
C	H

replaces

superior	inferior
C′	C
H′	H

A sample document fragment structured according to the DN in Fig 3 for the toaster T is shown below:

> T is a basic toaster that can detect when the user has pressed down the lever and start heating the toast for a set time. Once this time has passed, the heating is switched off and the lever returned to its original position. T is composed of two subcomponents: the heating element (H) and the control module (C). These will be described later in the document.

Similar fragments exist for all the components except C' and H'. However, this does not affect the documentation for T which will have the fragments in the order shown below:

fragment (T)
- fragment (C)
- fragment (H)
  - fragment(M)

The hierarchical structure is obtained by the *contains* relation and the sequence from *uses* relation.

Another build of T (T') is made but since component C is not available it is replaced by C'. Fragment(C') does not exist but since only the functionality of C will be realised in this build, the documentation can remain unchanged.

A third build is now made based on T' (T'') which has H' instead of H. However, this time the fragment C cannot be used to describe C' since the additional functionality can now be used because the heating element is able to deal with temperature (browning) changes. Hence, the documentation cannot be completed until fragment(C') and fragment(H') are ready.

With a data model as the one shown, it is possible to determine whether all the fragments are available to produce documentation for a given build. For a simple example like this toaster, this may seem trivial. However, for large systems with hundreds of components where the documentation is received from many sources, the searching of fragments and generation of documentation becomes correspondingly hard.

6. CONCLUSIONS AND FUTURE WORK

Previously, we have worked on architectures and software reuse [15, 16], and more recently on the structure of technical documentation [9, 17]. In this paper we have brought these two strands of research together.

As future work, we will investigate the relevance of this documentation model in different varieties of system evolution. So far we have only studied the case where the components in a system become progressively more advanced. Other changes include re-factoring the system functionality (logically related components can be grouped to form one, say) and the production of a family of products that are based on a common core [4]. Is it then the case that the author starts with a core document that is relevant to all the products and extends it to fit each product?

The data model in this paper has also been implemented so that we are able to carry out further experiments with real systems.

Just as software components are reused to increase productivity, document fragments should also be reused. However, traditional documentation does not lend itself very well to reuse [18]. In order to reuse a component, one has to understand its functionality and how it can be used in a specific context. We cater for this requirement by arguing that successful reuse can be achieved by defining a common structure, extracting common information and extending current documentation.

Producing high-quality documentation is a complex task. It should ideally parallel the development of the artefact [19] and can benefit from reflecting the structure of the system being described [20]. We have shown that there is a strong mapping between the system architecture and the way in which its documentation is composed and thought about. We believe this will improve the quality of both the architecture and the documentation, and increase the extent to which both can be reused.

REFERENCES

1. Clements, P., et al., *Documenting Software Architectures: Views and Beyond*. 2003: Pearson Education.
2. IEEE, *ANSI/IEEE Standard 1471-2000: Recommended practice for architectural description of software-intensive systems". (Available online at http://ieeexplore.ieee.org/servlet/opac?punumber=4278470; last accessed 5.3.2008)*.
3. Garlan, D. and M. Shaw, *An Introduction to Software Architecture.* Advances in Software Engineering and Knowledge Engineering, 1993. **1**.
4. Müller, J.K., *The Building Block Method: Component-based Architectural Design for Large Software-intensive Product Families*. 2003, Universiteit van Amsterdam.
5. Kruchten, P.B., *The 4+1 View Model of architecture.* Software, IEEE, 1995. **12**(6): p. 42-50.
6. Meusel, M., K. Czarnecki, and W. Köpf, *A model for structuring user documentation of object-oriented frameworks using patterns and hypertext* in *ECOOP'97 — Object-Oriented Programming*. 1997, Springer Berlin / Heidelberg.
7. Gatzemeier, F., *Patterns, Schemata, and Types—Author Support Through Formalized Experience*. 2000. p. 27–40.
8. Mann, W. and S. Thompson, *Rhetorical Structure Theory: Toward a functional theory of text organisation.* Text, 1988. **8**(3): p. 243-281.
9. De-Silva, N. and P. Henderson. *Narrative-based writing for coherent technical documents*. in *ACM Special Interest Group on the Design of Communication*. 2007. El Paso, Texas, USA.
10. Holt, R., *Binary Relational Algebra Applied to Software Architecture*, in *CSRI Tech Report 345*. 1996, University of Toronto, Canada.
11. Kruchten, P., *What do software architects do?*, in *Available online at http://www.sei.cmu.edu/architecture/what_architects_do.pdf*. 2006.
12. Soni, D., R.L. Nord, and C. Hofmeister. *Software architecture in industrial applications*. in *17th International Conference on Software Engineering (ICSE)*. 1995. Seattle, USA: ACM Press New York, NY, USA.
13. Minto, B., *The pyramid principle*. 3rd ed. 2002, UK: Pearson Education Limited.
14. Gregor, S., J. Hutson, and C. Oresky. *Storyboard Process to Assist in Requirements Verification and Adaptation to Capabilities Inherent in COTS*. in *First international conference on COTS-Based Software Systems (ICCBSS 2002)*. 2002. Orlando, USA: Springer.
15. Henderson, P. *Laws for Dynamic Systems*. in *International Conference on Software Re-Use (ICSR 98)*. 1998. Canada: IEEE Computer Society.
16. Henderson, P. and J. Yang. *Reusable Web Services*. in *8th International Conference on Software Reuse (ICSR 2004)*. 2004. Spain: IEEE Computer Society.
17. De-Silva, N., *A narrative-based collaborative writing tool for constructing coherent technical documents*, in *School of Electronics and Computer Science*. 2007, University of Southampton: Southampton, UK.
18. Sametinger, J. *Reuse documentation and documentation reuse*. in *TOOLS 19: Technology of Object-Oriented Languages and Systems*. 1996. Paris, France: Prentice Hall.
19. Priestley, M. and M.H. Utt, *A unified process for software and documentation development*, in *Proceedings of the 18th annual ACM international conference on Computer documentation: technology & teamwork* 2000, IEEE Educational Activities Department: Cambridge, Massachusetts. p. 221-238.
20. Sametinger, J., *Object-oriented documentation.* ACM Journal of Computer Documentation, 1994. **18**(1): p. 3-14.

A Software Framework for Integrative Physiological Model Simulation

E. Zeynep Erson and M. Cenk Çavuşoğlu

Department of Electrical Engineering and Computer Science, Case Western Reserve University
10900 Euclid Av., 44106-7221, Cleveland, OH,USA *

Abstract

Emergence of systems biology motivated more comprehensive and integrative approaches for modeling physiological processes. Presented study proposes a software framework to integrate multilevel and multiscale models for physiological processes. The aim of the study is to provide the interfacing mechanisms to facilitate the integration of models from multiple research groups increasing the ability to construct complex simulations of physiological models. In this paper, high level design of the proposed system, integration of physiological models and the architecture to modularize the integration are presented. As the proposed system targets a multiscale and multilevel integration of mathematical models, complex diseases or physiological processes effecting many organs or organ systems, like diabetes, are within in the application areas. Besides enhancing the model development processes; the presented study will accelerate the development, analysis and testing of integration approaches for multiscale and multilevel physiological models.

1. Introduction

Emergence of systems biology provided a comprehensive and integrative perspective to examine the structure and function at the cellular and organism levels instead of focusing on the isolated parts [6, 14]. However the challenge of building medical simulations where multiscale and multilevel physiological processes are developed together is often too great for any individual group since expertise from different fields is required. Therefore it is necessary to have frameworks where various models can be integrated leading to new simulation models from independently developed models. The present study addresses this challenge and proposes a software framework to integrate mathematical models of physiological processes ranging from intra cellular level up to organ, organ system and organism levels. Specifically; the aim is to facilitate the integration of multiscale and multilevel models of physiological processes in a modular framework. To achieve this task, instead of building the architecture based on the domain specific components such as anatomical and physiological information; we are focusing on the application enforced functionality, *integration of information*.

Mathematical models for the physiological processes represent the regulation, control and modification of a physiological variable which has an effect on defining the current state of the whole system [7]. A change in a physiological variable has a direct or indirect effect on processes determining other physiological variables. In other words, every physiological variable carries an information which needs to be accessed, used, modified or integrated by other variables. Therefore integration of physiological processes is conceptualized by the transfer, access or sharing of information among the models representing the processes, and will be referred as *information flow* throughout the paper.

In the proposed framework, models to be integrated are decoupled by separating the mechanisms of information flow from the information itself. Information flow architecture, which is a crucial part for the model integration is the focus for this paper.

In addition to the integration of various physiological processes, the software will also enable using different integration algorithms and approaches with the interfacing mechanism. Therefore developers will have control on what to integrate as well as on how to do the integration. With these attributes, the framework will provide a user friendly, plug-and-play type environment where both individual models of physiological processes and different integration approaches can be used.

In the next Section (Section 2), studies that are using integrative approaches and their attributes which motivated the proposed architecture are summarized. The rest of the presentation for the framework will be based on a case sce-

*This work was supported in part by National Science Foundation under grants CISE IIS-0222743, EIA-0329811, and CNS-0423253, and US DoC under grant TOP-39-60-04003.

nario using a circulatory system model [3] coupled with an IV drug model and its effects on a target organ [20]. In Section 3 details of the problem and the conceptualization of the system for integrating physiological processes is discussed. High level design of the proposed solution and overview about the system components are presented in Section 4. Design of the information flow idea, being the focus of this paper is discussed in Section 5 accompanied with the implications on the case scenario. Implementation details for the current status of the project are given in Section 6 to show how the components, which are not discussed in detail in this paper are integrated with the information flow mechanism.

2. Background

Integrating multilevel physiological processes requires both structural and functional hierarchical information for the contributing models. The hierarchical structure of the anatomy represents the organization starting from DNA sequences, RNA and protein, protein-protein interactions and protein-DNA interactions, to cells, tissues, organs, organ systems to the whole organism [6]. Modularity concept is also expanded for functionality in biological systems [10]. Importance of modularization is realized more as the multiscale modeling came into consideration for integrative physiology studies as modularization simplifies multiscale modeling [15]. Therefore in the proposed software framework, anatomical and physiological knowledge is defined using a modular, systematic representation.

One of the successful studies in cell level modeling, with an integrative approach, is the BioSPICE Project, which provides a framework for modeling, simulating intra and inter-cell processes. BioSPICE project also provides an integrative software environment that enables access to different computational biological tools [11].

Physiome Project [3], has a database of physiological models with different scale and levels. With the hierarchy of models from cell level to organ level, the project aims to analyse integrative biological function models and test the hypothesis using mathematical models [12]. JSIM [1], which is a Java-based system, is used to simulate the models in Physiome model repository. Although the models to be simulated vary from cellular level to organ and system level, they can only be simulated independently.

With the emergence of systems biology, development of modeling and simulation tools for this domain increased, such as, SCIRun [5] and Systems Biology Toolbox for Matlab [19]. Although SCIRun is a general purpose problem solving environment for physical and biological systems, it does not provide a simulation and modeling framework. It uses a data-flow architecture to integrate the steps of preparing, executing, and visualizing simulations of physical and

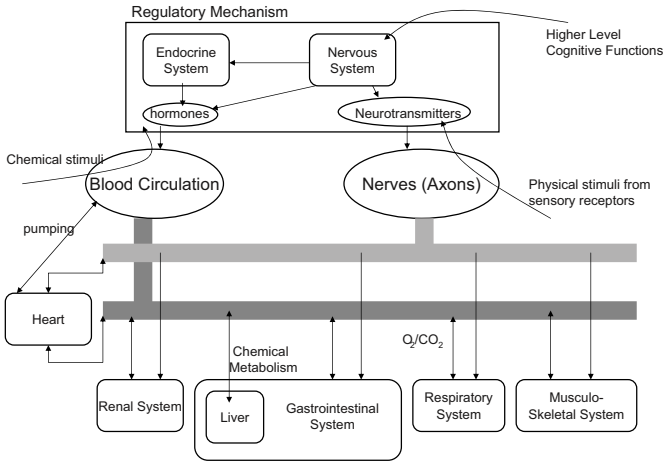


Figure 1. Information flow among physiological processes through circulatory and nervous systems.

biological systems. The Systems Biology Toolbox for Matlab provides an extensible environment for modeling, simulation, importing SBML (Systems Biology Markup Language) models and analysis tools.

Modeling and simulation of complex physical systems have been extensively studied outside the biology domain. There are tools and languages, such as, Modelica [4], Matlab Simulink [2] and Ptolemy [9] that provide creation and simulation of mathematical models for physical systems as well as integration of submodels. However none of these tools or languages are designed for the specific domain of biological systems.

3. Problem Specification

Every physiological property (blood pressure, blood glucose level, body temperature etc.) is associated with an anatomical structure; and the mechanism that controls, modifies and regulates is represented with a physiological process [8]. Circulatory and nervous systems are the mechanisms that manage the flow of information among the processes and physiological properties. The flow of information in the circulatory system can be thought of as a broadcasting mechanism, where information in the form of physiological variables are transported in the blood stream. On the other hand, the nervous system can thought as a point-to-point communication mechanism where the information in the form of electrical signals are transmitted (See Figure 1). Once the information is disseminated among the processes, individual models representing the processes integrate the available information.

A case scenario is used to present the proposed solu-

tion for the problem of handling information flow among physiological processes and integration of the information. In the presented case, concentration of an intravenous(IV) drug is the information to be carried through the circulatory system. Cardiopulmonary mechanics model from the Physiome Project Model repository [3] is used to model the circulatory system. The cardiopulmonary mechanics model is composed of a four-chamber varying-elastance heart, pericardium, systemic circulation, pulmonary circulation, coronary circulation, baroreceptors, and airway mechanics. Model for the IV drug represents the changes in the concentration of the injected drug in the injection site, vascular mixing, concentration in the arterial tree and concentration at the target organ [20].

4. High Level Design of the System

As seen in Figure 2, a layered design separating the structural and functional information from the information flow mechanism is proposed. The dependency among the layers are in one direction keeping the coupling among separate layers low. The design decision for separating the anatomical and physiological ontology and functionality, has an advantage for the reusability and extendability of the framework. The developers will be getting advantage of a higher level of reuse, which is an important advantage of using ontology based architecture [21].

Mathematical models are used to represent the dynamics of the physiological attributes. Mathematical representation of any processes is independent of the physiological variable that it controls, regulates or modifies. However every biological process depends on the mathematical model as it has different concurrency and time constraints. While some processes occur at discrete time steps and can be described by algebraic equations, some processes span a continuous time frame and can be described with differential equations. On the other hand some processes may not show a regular behavior, and can occur at specific times. Models of computation can be thought of as design patterns in object oriented paradigm, which will behave as the core of the solution [13]. Based on this analogy, the models of computation are modularized in *Computational Layer* and are classified according to the ways they deal with concurrency and time concepts, as: *Continuous Time Models*, *Discrete Time Models*, *Discrete Event Models*.

In order to have a modular representation of physiological processes and variables, a high level ontology is developed. Physiological processes are defined based on their qualitative, quantitative and temporal attributes. The systematic representation of the physiological information is modularized in *Physiological Layer*.

Anatomical associations of the processes are defined with the hierarchy represented through the anatomical ontology. Foundational Model of Anatomy, FMA [16, 17, 18], is used to represent the taxonomy and part-whole relations for the anatomical information. Ontological representation of the anatomical information is defined in *Anatomical Layer* and is independent from all other layers.

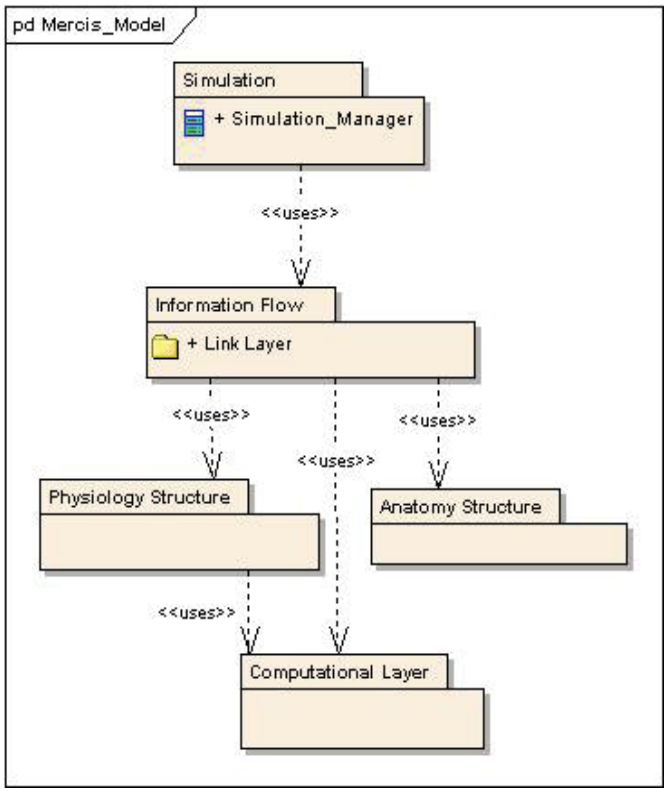


Figure 2. High Level Design

In the architecture shown in Figure 2,link layer handles the flow of information and integration of the information uses the anatomical and physiological information from the lower levels (See Section 5). Simulation of the integrated models is managed by the *Simulation Layer*, behaving as an application layer. In the following section details for the *Link Layer* is given within the realm of information flow and information integration.

5. Information Flow

Link Layer which sits on top of the physiological and anatomical layers, is designed as in Figure 3. Structural and functional information is encapsulated in *Component*s, which correspond to a single physiological variable and a corresponding model. A number of components come together to represent a physiological process. In the presented sample models, cardiopulmonary mechanics is composed of 182 *component*s each of which correspond to a physiological property with a mathematical model. The depen-

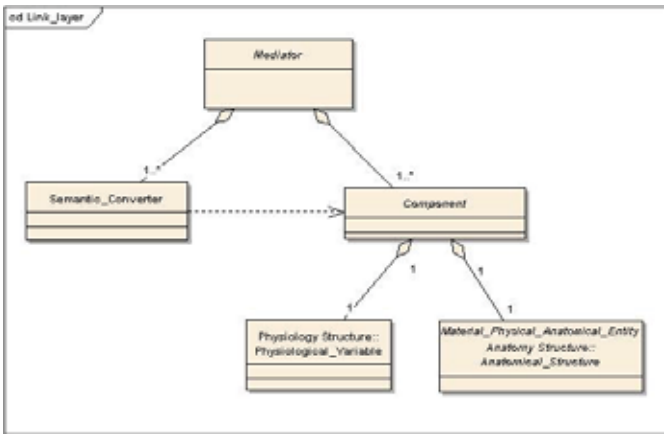


Figure 3. Design for the layer responsible of the information flow among the individual physiological models.

dency among physiological variables are defined through *Semantic Converters*. These units decouple physiological variables by encapsulating the semantics of the dependency. In the case of the cardiopulmonary mechanics model, mathematical model regulating the cardiac output depends on the values of the flow through the aortic valve which depends on other variables. Such dependencies among variables are also used to determine the simulation order. Therefore the semantics of the dependency among the *Components* is defined in the *Semantic converter*s and is used by the *Mediator* to pass to the upper layer, *Simulation Layer*.

Mediator is responsible for compiling the integrated models to handle mediation of information, results of which will be passed to the simulation layer to run the models. In order to have a serializable simulation order of components, the dependencies among components should be resolved. The *Mediator*, resolves the cyclic dependencies in the compilation process. Except for the algebraic loops, *Mediator* will resolve the cycles and create a sequential order of components using the dependencies defined by the *Semantic Converters*.

For the case of the circulatory system, information flow idea presented above is extended. Components that are part of the circulatory system are grouped as *Extrinsic* and *Intrinsic*. Intrinsic components correspond to the physiological variables and models that determine the mechanics of the flow of information, such as the cardiac output, flow of blood at the arterial tree, blood pressure, etc. Extrinsic components correspond to variables representing the information carried through the blood stream, using the intrinsic information. In the presented case, cardiopulmonary mechanics model constitutes as the intrinsic model. Model determining the concentration of the IV drug corresponds to the extrinsic model. Therefore, components in the IV drug model integrates the information from the cardiopulmonary mechanics model and transports the information about the concentration of the injected drug in to the circulatory system. These two models are integrated over the cardiac output variable which is defined as an intrinsic component. If the IV drug model were to be simulated as is, the cardiac output variable will be a constant and its regulations, changes will not be considered. By integrating the cardiopulmonary mechanics model with the IV drug model as an integration of intrinsic and extrinsic models, we were able to see the effect of change in cardiac output on the drug concentration in the blood stream.

Two types of integration schemes are used in the proposed system. The presented case of integration of information through the circulatory system, is an example of *horizontal* integration. Horizontal integration refers to the flow of information and integration among the physiological variables which have the same level anatomical structure associations. In the presented case scenario,the variables for the cardiac output in the IV drug model and the cardiopulmonary mechanics model were horizontally integrated. The integration mechanism replaces the constant representation of one variable in IV drug model with the regulated variable in cardiopulmonary mechanics model. Anatomical information associated with the physiological processes guide the integration process and determine the choice for the semantics of the integration. In the IV drug model, the cardiac output, defined as the blood flow through heart is an attribute associated with the heart. The variable to be integrated on the cardiopulmonary mechanics model is associated with an anatomical structure, the same structure in this case, which is in the same level in the anatomical ontology representation.

The other type of integration is the *vertical* integration. Multiscale and multilevel integration of physiological processes will be handled with this type of integration mechanisms. In the case of multilevel integration, the variables to be integrated are associated with anatomical structures having part-whole or parent-child relationships. Semantic converters will implement the vertical integration approaches. Aggregation and dispersion of the variables are the basic approaches proposed for implementing semantics of the vertical integration of multilevel models. For the case of the multiscale integration, the real challenge is at the computational side. There are two approaches to simulate multiscale models. The first approach is a brute force technique, and relies on simulation of all the individual (low level) subcomponents to aggregate and compute the high level behavior. This type of aggregation is naturally supported by the object-oriented design, which allows hierarchical construction. However there is a practical limitation of this approach, the computational cost. The second ap-

proach tries to reduce the computational complexity by relying on model reduction techniques. Since mathematical models for complex biological systems both contain linear and nonlinear models, an approach handling both of these systems should be adopted. Model reduction for large scale nonlinear dynamical systems is an open problem and is outside the scope of this paper.

6. Implementation Details

Current implementation of the proposed system uses models from the Physiome Project repository. Physiological processes in this repository systematically describes models from the literature using the Mathematical Modeling Language (MML). Presented system preprocesses MML models to create the library of mathematical models to be used by the physiological processes. MML files are parsed on line to create *Component*s with the physiological variables. On line parsing also creates *Semantic Converters* extracting the dependencies among the physiological variables from the model equations. Model developers can choose to associate these components with anatomical structures using the ontology representation in the framework.

In Figure 4, a prototype is presented to build the integrated system for the aforementioned system with the cardiopulmonary mechanics and the IV drug model. The first step is to build the medium for the flow of information, circulatory system. As stated in Section 5, models contributing to the mechanics of the circulatory system are modeled as *intrinsic* models. Having defined the intrinsic variables for the circulatory system, the second step is to add the extrinsic information to the model. Users can load the selected .mml file and add the information to the circulatory system by declaring the model as an *extrinsic* model. Third step is presented to show how the information flow mechanism can be used to access the variables in the circulatory system. In Figure 4, the third model loaded is the part of the IV drug model which calculates the effect of the drug at a target organ. This model accesses both the intrinsic variables, such as the blood flow and extrinsic variables like the drug concentration in the blood stream. Although the dependencies within a single model are extracted automatically by the parser, the points of integration for the loaded models should be user controlled. The last step handles the horizontal integration among the user defined integration points, which are the cardiac output and concentration of the drug in arterial system for the presented case. Having the required information to compile the model to prepare for the simulation, *Mediator* compiles the models, performs the integration, and passes the required information to the simulation step.

7. Conclusion

In addition to providing an integrative simulation environment for complex biological systems, the presented architecture will facilitate shared model development as well as data and model sharing among multiple research groups. Therefore the proposed framework will bring a new perspective with the multiscale, multilevel model integration approach.

Although major components of the system are complete, the development step is being pursued in the context of possible applications. As we are targeting a multiscale and multilevel integration of mathematical models, diseases or physiological processes effecting many organs or organ systems are within in the application areas. Diabetes, which has complications such as heart diseases, blindness, nerve damage and kidney damage, is one of the most interesting application areas, having effects on many organs and organ systems. Another complex process which presents an application area for the proposed framework is Orthostatic Tolerance. It is a very critical measure for astronauts or anyone who faces sudden gravitations changes and gravitational stress. Orthostatic tolerance is dependent on the degree of vasoconstriction and the magnitude of plasma volume, which in turn determines the tendency to faint when standing upright. This mechanism, having effects on circulatory system and nervous system is another very interesting application area to integrate different models in our software framework.

References

[1] Jsim, http://www.physiome.org/jsim/.

[2] Mathworks Inc., Simulink, http://www.mathworks.com/products/simulink.

[3] Physiome project, http://www.physiome.org/.

[4] Modelica, a unified object-oriented language for physical systems modeling; language specification 2.0. the modelica association, http://www.modelica.org, 2002.

[5] SCIRun: A scientific computing problem solving environment. scientific computing and imaging institute (sci), http://software.sci.utah.edu/scirun.html, 2002.

[6] A. Aderem. Systems biology: Its practice and challenges. *Cell*, 121:511–513, 2005.

[7] R. M. Berne and M. N. Levy. *Physiology*. Mosby, 1998.

[8] R. M. Berne and M. N. Levy. *Principles of Physiology*. Mosby, 2000.

[9] J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *International Journal of Computer Simulation special issue on Simulation Software Development*, 4, 1994.

[10] M. E. Csete and J. C. Doyle. Reverse engineering of biological complexity. *Science*, 295:1664–1669, 2002.

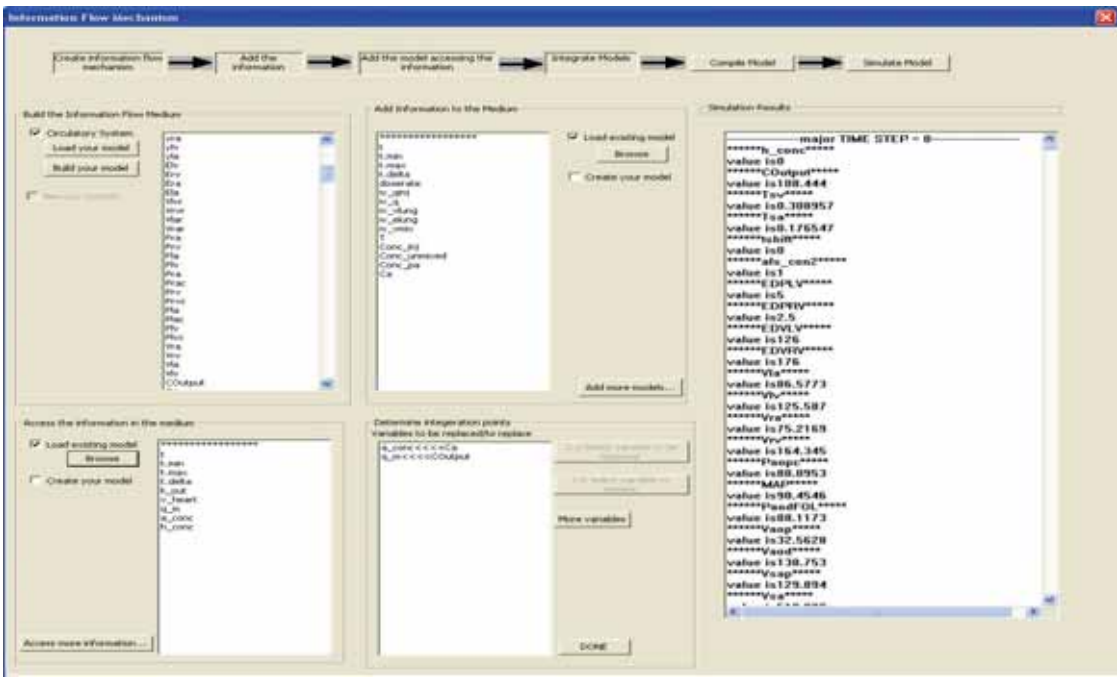


Figure 4. Prototype showing the integration of models to create the circulatory system's extrinsic and intrinsic components.

[11] T. Garvey, L. P., C. Pedersen, D. Martin, and M. Johnson. Biospice: access to the most current computational tools for biologists. *Omics: A Journal of Integrative Biology*, 7:411–420, 2003.

[12] P. J. Hunter and T. K. Borg. Integration from proteins to organs: the physiome project. *Nature Reviews, Molecular Cell Biology*, 2003.

[13] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng. Overview of the ptolemy project. Technical report, Department of Electrical Engineering and Computer Science, University of California, Berkley, 2003.

[14] H. Kitano. Systems biology: A berief overview. *Science*, 295:1662–1664, 2002.

[15] D. Mackenzie. Ramping up to multiscale-taking biomedical modeling to a new level. *Biomedical Computation Review*, 2006.

[16] P. J. Neal, L. G. Shapiro, and C. Rosse. The digital anatomist spatial abstraction: A scheme for the spatial description of anatomical features. In *Proceedings, American Medical Informatics Association Fall Symposium*, pages 423–427, 1998.

[17] C. Rosse, J. L. V. Mejino, B. R. Modayur, R. M. Jakobovits, K. P. Hinshaw, and J. F. Brinkley. Motivation and organizational principles for anatomical knowledge representation: The digital anatomist symbolic knowledge base. *Journal of the American Medical Informatics Association*, 1998.

[18] C. Rosse, L. G. Shapiro, and J. F. Brinkley. The digital anatomist foundational model: Principles for defining and structuring its concept domain. In *Proceedings, American Medical Informatics Association Fall Symposium*, pages 820–824, 1998.

[19] H. Schmidt and M. Jirstrand. Systems biology toolbox for matlab: A computational platform for research in systems biology. *Bioinformatics Advance Access*, 22, 2005.

[20] R. Upton. A model of the first pass passage of drugs from i.v. injection site to the heart-parameter estimates for lignocaine in the sheep. *British Journal of Anasthesia*, 1996.

[21] X. Wang, C. W. Chan, and H. Hamilton. Design of knowledge-based systems with the ontology-domain-system approach. In *Proceedings of SEKE*, pages 15–19, 2002.

Combining SOA and BPM Technologies for Cross-System Process Automation

S. Herr[1], K. Läufer[2], J. Shafaee[2], G. K. Thiruvathukal[2], G. Wirtz[1]

[1]Distributed and Mobile Systems Group, University of Bamberg

Feldkirchenstraße 21, 96052 Bamberg, Germany, sebastian.herr@gmail.com, guido.wirtz@uni-bamberg.de

[2]Emerging Technologies Laboratory, Dept. of Computer Science, Loyola University Chicago

820 N. Michigan Avenue, Chicago, Illinois 60611, U.S., {laufer|shafaee|gkt}@cs.luc.edu

Abstract

This paper summarizes the results of an industry case study that introduced a cross-system business process automation solution based on a combination of SOA and BPM standard technologies (i.e., BPMN, BPEL, WSDL). Besides discussing major weaknesses of the existing, custom-built, solution and comparing them against experiences with the developed prototype, the paper presents a course of action for transforming the current solution into the proposed solution. This includes a general approach, consisting of four distinct steps, as well as specific action items that are to be performed for every step. The discussion also covers language and tool support and challenges arising from the transformation.

Keywords: SOA, BPM, BPMN, BPEL, WSDL, standards, application integration, BPIOAI, web services

1. Introduction

As part of their efforts to automate enterprise-wide business processes, organizations are often faced with the challenge of integrating the data and business logic of several independent application silos [6]. This issue is traditionally addressed through custom-made solutions that are expensive to build and maintain, inflexible to changing requirements, error-prone and often poorly aligned with the enterprise's business goals. One promising and increasingly recognized approach to this problem is to combine Business Process Management (BPM) and Service Oriented Architecture (SOA) concepts and technologies with the goal of forging a flexible and cost-efficient process automation and system integration solution. In combination, both paradigms appear to be of significant benefit to each other. BPM's lack of focus on architectural principles (e.g., loose coupling, service reusability) and its poor flexibility in regard to technology choice (i.e., vendor lock) are addressed by SOA. SOA on the other hand can benefit from BPM's top-down, requirements-oriented, and visualization-focused approach that considers the entire life cycle of business processes.

Despite the general recognition of a SOA and BPM convergence and the resulting synergy [3, 7, 9], there is still an extensive lack of best-practice examples and real-life industry adoption, in particular in medium-sized and small enterprises. One reason for this is a shortage of resources paired with the common believe that SOA and BPM initiatives only bring return on investment (ROI) if applied on a large scale. While this assumption may hold some cases, we argue that using the suggested paradigms results, even in small-scale scenarios, in considerable benefits that have the potential of significantly outweighing the anticipated costs.

To back this proposition, we conducted an industry case study with the goal of applying a combination of SOA and BPM concepts and technologies as a replacement for a custom-made, proprietary process automation solution. Our focus was the universal use of standards, specifically the Business Process Modeling Notation (BPMN) and the Business Process Execution Language (BPEL). In particular, we identified weaknesses of the currently deployed solution and specified a transformation of this solution to the envisioned replacement, including a roadmap with specific required action items along with the selection of pertinent tools. Finally, to address financial considerations as well as several other concerns, we thoroughly evaluated and compared both solutions, with special emphasis on addressing the previously identified weaknesses. The study was conducted in the setting of a top-five global hosting company with the intent of delivering a proof of concept for the feasibility and impact of the proposed approach. The transformation itself was realized in the form of a prototype (PT) that included process-models (BPMN) and -code (BPEL), service interfaces (WSDL) and "dummy" implementations of services, but no end-to-end implementation with the actual enterprise applications.

Our work was influenced by recent publications addressing the SOA/BPM convergence (e.g., [3, 7, 9]) as well as general literature on approaches to application integration [5] and SOA [1, 4, 6]. During the PT development, we have relied extensively on the BPMN-to-BPEL mapping rules [8].

The remainder of this paper is organized as follows. In section 2, after briefly describing the existing solution and its weaknesses, we introduce the chosen pilot process. In section 3, we present the approach and results of the transformation, while section 4 compares both solutions and discusses how the issues from section 2 are addressed. We conclude with a summary and a discussion of future work.

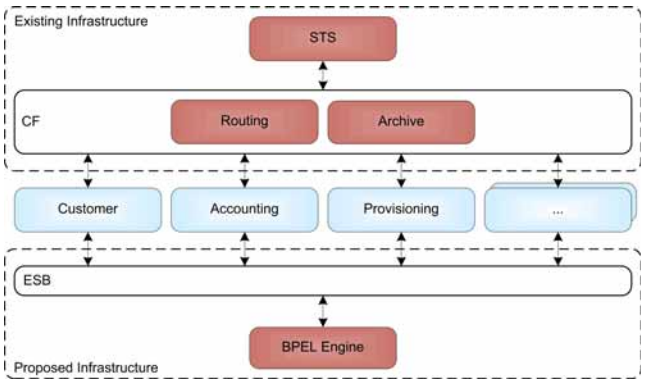


Figure 1. Enterprise Architecture

2. Initial Situation

The environment's system landscape comprises ten physically separated back-office applications that each take over a distinct business or auxiliary function (e.g., Accounting System, cf. Fig. 1). Together, these systems fully automate most of the company's business processes including sales, billing, and provisioning, to name a few.

2.1. Existing Process Automation Solution

The company's process automation approach integrates a custom-made state transition system (STS) for process state management and business rule enforcement with a custom-made communication framework (CF) for system integration (cf. Fig. 1 top). Information exchange between back-office systems is realized via asynchronous XML-based messaging. Typically, a new process is brought to life inside the STS, which is triggered by an external event such as the submission of a new sales order. The STS manages the process execution by sequentially starting various integration flows via the CF. Integration flows commonly visit several back-office systems (sequentially and without the involvement of centralized coordinator), cause business logic updates and/or collect data, which is attached to the message and used later inside a different system as part of the overall process. Each integration flow eventually returns to the STS causing a process status update, which results in further actions.

2.2. Solution Benchmark

We have identified four high-level issues: initial development and setup costs, maintenance costs (i.e., integration of new systems, implementation of new business requirements, etc.), technology and solution learning curve and the likelihood of introducing errors during the mapping of business requirements to process implementations. Upon examining these issues, we have detected four problems as their respective root causes. In turn, these problems give rise to the criteria for the evaluation and comparison with the suggested solution.

1. *Lack of Sufficient and Practical Documentation*

Acquiring sufficient knowledge to understand and work with the existing solution requires collecting and aggregating information from various sources (e.g., people, documents, code) and was experienced as a highly time-consuming part of the case study. Also, the poor documentation increases the likelihood of introducing errors. Besides a lack of language and technology tutorials, which can be ascribed to the proprietary nature of the solution, the environment also misses a common communication basis in the form of a process model that can be shared among business analysts, architects and programmers. Without such a model, new requirements are likely to be misinterpreted when handed to a programmer. Finally, even if sufficient documentation in the form of state-transition (ST) tables, ST diagrams and flow diagrams were available, it would not provide a complete and practical picture.

2. *Complexity of Solution and Process Setup*

First, the initial setup required building most components from scratch. This includes the STS with database setup, database triggers, stored procedures, the event module as well as the CF with its routing and archive modules, message parser but also the development of the message structure (i.e., routing information) and other rules for new process setup. Merely the message queues (MQ) (one per system) could be acquired from a third-party vendor. Second, new processes or even slight adjustments in existing processes require careful and complicated planning and design. States, state transitions and events need to be designed and set up. Routing information has to be added or changed, integration flows and additional message parsers have to be implemented. This high effort of process maintenance is even further intensified by the solution's poor separation of concern (SoC). Although process implementations are distributed across two components (i.e., STS and CF), the solution places some of the process-specific behavior inside the back-office systems (i.e., in some instances, the route of an integration flow is changed during system invocation). This again requires most systems to be aware that they are part of a higher-level process and hence prevents them from being reused by other processes.

3. *Poor Deployment and Testing Conditions*

Process deployment requires manual addition or replacement of routing information, states, events, etc. "One-click" deployment is not supported. The effort for process testing has been indicated by responsible personnel as being extremely high. First, the various steps of a process (i.e., the systems invocations) cannot be tested individually but only as part of an integration flow. Second, process debugging is only possible in retrospect by evaluating the log files of completed flows. Third, the solution does not allow for design-time code validation through respective tools.

4. *Degree of Business-IT Alignment*

The solution is clearly deficient with respect to the alignment of the enterprise's processes with its IT infrastructure. The nature of the solution generally makes it difficult to translate business requirements into implementations. Every process, coming as a whole from the business side, needs to be split up into several flows that are tied together via the STS. It

is often difficult to decide which parts of the process should be realized through flows and which parts require state and corresponding state transitions. Also, the question arises whether process-specific rules should be enforced inside the STS or inside the pertinent back-office system. In addition, the implementer must sometimes deviate from the order of steps provided by the business analyst.

2.3. Selected Business Scenario

As the basis for our PT, we chose the company's sales process as the pilot to be implemented with the suggested approach. We picked the sales process because of its universality (i.e., industry neutrality), its size and complexity (involving all of the organization's back-office systems), and its volatile nature that would emphasize the benefits of the proposed solution. At a glance, after an order has been placed via an online shopping cart environment, the back-end process is started performing various steps including the creation of a customer account, a fraud check, the processing of financial transactions and the initiation of product provisioning. Each step is implemented by a different back-office system.

3. Solution Transformation

After analyzing the existing process automation solution and taking a closer look at the selected pilot process, it became clear that a simple transformation to the suggested technologies and paradigms would be insufficient to best demonstrate their benefits. First, a mere transformation without addressing the poor SoC of the current setup including its weak service reusability potential did not appear to be meaningful or at least would not allow us to demonstrate the potential for reusability in the long run. Second, during the reverse-engineering effort of the sales process from source code and residual process documentation, we were able to discover several crucial errors in process logic that could easily be repaired with the new approach. Finally, the current solution does not implement process-wide transactional behavior; this issue could also be addressed effortlessly and hence was added to the list of objectives.

To reduce the complexity of the transformation process, we decided to address the various goals and objectives in four sequential steps. This so-called "transformation lifecycle" (Fig. 2) can be reused for porting additional business processes in the future. Phase one sought to integrate the existing systems using BPEL. This implies exposing the respective functionality via web services (WS). Additionally, the issues of SoC and reusability have been addressed by shifting some functionality between systems and by applying well-deliberate service design. Phase two aimed to bridge the gap to the business side by providing a visual, more business-oriented representation of the sales process using BPMN. Based on the newly created models from phase two, phase three directly repaired process errors and added business transactions from a requirements-oriented standpoint. Finally, phase four synchronized the adapted process models with the previously created implementation. Each of the four

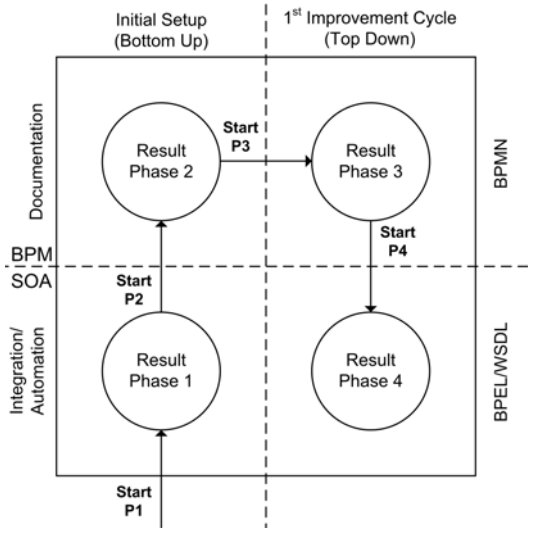


Figure 2. Transformation Life Cycle

steps will be discussed in detail below.

Phase 1: Bottom-Up Process Automation

In phase one BPEL v1.1 and WSDL v1.1 where used with Oracle's JDeveloper tool that supports various SOA-related development tasks including service-stub generation from WSDL-files and visual BPEL modeling.

First, we exposed the existing systems as WS. As a result, 100% of the systems' functionality (i.e., business logic) was reused and system access has merely been transformed from the CF to WSDL-based access. One important task was the definition of service inputs and outputs: business documents, such as an invoice, that are specified through XML schemas. The challenge was to transform the structure of the existing XML message into various modular schemas that can each be passed into the respective service, individually or in aggregated form. The transformation was required for adopting BPEL's orchestration approach. A BPEL process locally stores process data, such as a sales order (consisting of a quote, customer data, product selection, etc.), and extracts data (e.g., one specific product item) in the course of a service invocation as needed. A second challenge was to improve SoC and potential for reusability. To this end, we moved some functionality between systems as well as from the systems into the process. The newly created services have been designed accordingly (i.e., assuming the functionality in the new place).

Second, we implemented the sales process in BPEL by orchestrating the previously designed services. This required a transformation of the state- and flow-based solution into an activity-based solution. Process state management and routing of messages is now realized transparently by the respective infrastructure, and the various integration flows were combined into one process. Our experience was that all aspects of the sales process could be expressed in BPEL with no restrictions.

As our deployment and testing environment, we installed the Oracle SOA Suite. It ships with an Enterprise Service

Bus (ESB) and a BPEL engine and integrates nicely with the chosen IDE (JDeveloper), allowing BPEL processes and services to be deployed from within the IDE. The installation of the BPEL engine and ESB, or optionally a lightweight WS framework, makes the STS and CF obsolete (cf. Fig. 1). With some careful planning, it should also be possible to run both solutions concurrently until all processes have been ported.

Phase 2: Bottom-Up Process Visualization

The potential of the proposed solution can be fully utilized only with the addition of a process model that may be used as a communication basis among participating stakeholders. In phase two, we performed a mapping from the created BPEL code to a visual BPMN-based process model using BPMN v1.0. The model was created with the Microsoft Visio stencil extension from ITP Commerce. The resulting business process diagram (BPD) is illustrated in Fig. 3 (exclusive the highlighted parts). The mapping was done based on the rules by [8] and turned out to be very straightforward. All aspects of the BPEL code could be transformed to the BPD.

Phase 3: Top-Down Process Adaptation

The objective of phase three was to repair flaws, to add additional functionality and to make the process transaction-safe. Tool and language support is identical to phase two. As a first task, we swapped some steps in the process, which were out of order in the existing implementation. Secondly, we added one extra step at the end (i.e., "Send Order Completed Notification"), which was missing in the original solution. Thirdly, we extended the BPD with a manual order verification option and finally added rollback processes that would reverse prior system updates in case of process failure. The resulting artifact is an extended version of the BPD from phase two (i.e., Fig. 3, inclusive the high-lighted parts) as well as additional BPDs for rollback- and related processes (not shown here). It is important to recognize that phase three is entirely requirements-driven and independent of the underlying process and system implementations. Nevertheless, the new functionality was modeled based on design decisions that determined which system had to implement the new requirements in the future.

Phase 4: Top-Down Process Implementation

The final step in our life cycle was the synchronization of the BPDs from phase three with the existing implementation from phase one. Tool and language support are the same as in phase one. Most parts of the BPDs were implemented with no effort by using the visual process modeler of the IDE.

The implementation of the manual verification step and the transactional behavior is of interest. The latter was necessary to map the rollback processes from the BPD to the BPEL implementation. BPEL realizes loosely-coupled business transactions with its built-in compensation and fault handlers. A very convenient BPEL feature is the automatic execution of compensation handlers of those scopes that are already completed when the error occurs. An example for our case is the deletion of the sales order in the Sales System and the reversal of the "Create Customer Account"-sub process (e.g., after an order expired while waiting for manual payment, cf. Fig. 3). Both steps required an extension of the previously created WSDL interfaces with additional operations that simply reverse the existing operations.

The manual verification of orders naturally requires human involvement. Unfortunately, BPEL does not support human interaction in a standardized way. On the other side, we purposely refrained from using Oracle's proprietary BPEL extension for human workflows to avoid vendor lock. We rather addressed this issue by simply hiding the human involvement behind a custom-made service. In this case, the process asynchronously submits the sales order to an "Order Escalation Service," which prompts the need for order-verification to a user interface. The process waits in the current status until it is called back by the service after the issues has been resolved.

4. Evaluation and Solution Benchmark

This section benchmarks the existing solution against the proposed solution. We will discuss the issues identified in section 2.2 and outline how they are addressed in the new solution. We also provide some specific figures (i.e., cost savings) based on our experiences with the PT development. Nevertheless, the results presented here are conjectures that have not been measured owing to time restrictions. Determining the actual ROI would require operation and observation of the new solution in production environment over a period of several months.

In summary, the proposed approach combines state-transitions and integration flows into one artifact (i.e., BPEL process) whereas the BPEL engine orchestrates the various services into one enterprise-wide process. Changes in routes are now handled inside the process. Messages are transmitted per service invocation and not per flow. Service reusability is improved and process-wide transactional behavior is provided. The solution comprises an end-to-end process model clearly displaying all business requirements. Finally, the STS and CF are replaced by SOA and BPM infrastructure components (cf. Fig. 1).

1. *Documentation*

With the BPD (cf. Fig. 3) most information about the sales process and participating services is available at a glance. Aggregation of information from several sources (including source code review) is not necessary. The process clearly visualizes all steps, service invocations, possible faults, events, branches and business rules (e.g., expected values for decision points). The translation from the model to the implementation is unambiguous and the error likelihood is reduced. Since BPMN is specifically designed to be stakeholder neutral [8], it allows all participants to understand the model quickly. By contrast, ST- and flow diagrams may be harder to understand by purely business-focused staff. Furthermore, new stakeholders can quickly acquire

information about languages (i.e., BPEL, BPMN, WSDL) and infrastructure (i.e., BPEL engine, ESB) used in the project by studying some of the myriads of available tutorials and code examples. It will also be easier to recruit new stakeholders (e.g., engineers) that already carry the respective knowledge; this is not possible with a proprietary solution.

2. *Complexity of Solution and Process Setup*

The proposed solution has the potential of significantly reducing time-to market of the initial setup and of new requirements. Infrastructure is entirely replaced by third-party offers, which greatly reduces the setup time of the initial solution. Design and implementation of the generic parts of the STS and CF was indicated by the responsible manager with approximately US$70,000 (including purchase of several MQ licenses), which is already above the purchase price of a commercial SOA Suite license (US$50-65,000 assuming that one license will be sufficient). Nevertheless, despite the possibility of relying on open source infrastructure, we argue that even higher investments will pay off as the maintenance costs of enterprise applications commonly are much more significant [2] and this is where the suggested solution scores big points. Setup of new processes and adjustments in existing processes is far less time consuming. This is besides the improved testing conditions especially also ascribed to BPEL's predefined language constructs for process behavior as well as the tool support with its visual BPEL modeler, code generation etc. Furthermore, services can be designed for reuse, which again may reduce development time in the future. In particular, we anticipated savings of approximately US$10,000 in personnel cost only for the initial setup of the sales process. Simple adjustments in process logic can save up to 80% of implementation and testing time. With more complicated adjustments (e.g., the addition of a new payment option), this is even more significant.

3. *Deployment and Testing Conditions*

The solution's testing and deployment conditions have improved significantly. The infrastructure ships with built-in features that facilitate testing and debugging. Processes can be deployed effortlessly via the respective user interface or directly from within the IDE. Versioning is supported transparently (i.e., changed processes are deployed under a new version, running processes are completed in their old version). Furthermore, pre-deployment code verification is provided by the IDE and greatly reduces runtime errors. Finally, all services can be tested individually before attaching them to a process. The improved testing and deployment situation has a significant impact on the cost savings already discussed.

4. *Degree of Business-IT Alignment*

BPEL's orchestration approach in conjunction with the BPDs improves the solution's general degree of business-IT alignment. The process is visualized and implemented in a more natural way. Process behavior (e.g., branching, business rules, etc.) is integrated into one artifact and not distributed across several components. The number of steps in the process that cannot directly be mapped to a business requirement is reduced, thus, the gap between requirements and implementation is smaller.

5. Conclusion and Future Work

In this paper, we argued that a cross-system process automation solution leveraging SOA and BPM technologies can be significantly superior to a custom-built solution. We also presented an approach for porting the existing processes to the suggested solution. The study showed how the chosen concepts and technologies can be applied to a real-life scenario. All aspects of the pilot project (i.e. process model and implementation, services, etc.) were realized with the selected languages, and the PT was successfully tested. In conclusion, the study was a considerable success and demonstrated the merits of using a combination of SOA and BPM for process automation and system integration purposes. The resulting value gain was greatly recognized by the responsible personnel in the chosen environment.

Finally, to guarantee feasibility and long-term success, we recommend performing a more thorough analysis and evaluation of infrastructure and tool support for the specifics of the environment, including, for instance, issues such as tool selection. Second, an in-depth evaluation of quality of service aspects (e.g., scalability, security, reliability) should be performed. Furthermore, it may be advisable to extend the PT with an end-to-end implementation that involves the pertinent systems. As a last step, we believe that drafting an adoption strategy (i.e., guidelines, education of staff, adoption schedule) will help to discover other possible issues.

In general, the transformation life-cycle introduced in section 3 requires more case studies in order to adjust and generalize the porting process outlined in Fig. 2 in a way that it becomes useful also under different settings.

References

[1] N. Bieberstein, S. Bose, M. Fiammante, K. Jones, and R. Shah. *Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap*. IBM Press, 2005.

[2] L. Erlikh. Leveraging legacy system dollars for e-business. *IT Pro*, 2(3):17–23, May/June 2000.

[3] F. Kamoun. A roadmap towards the convergence of business process management and service oriented architecture. *Ubiquity Volume 8 , Issue 14 April 2007.*

[4] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA Service-Oriented Architecture Best Practice*. Prentice Hall Ptr, 2004.

[5] D. S. Linthicum. *Next Generation Application Integration: From Simple Information to Web Services*. Addison-Wesley Professional, 2003.

[6] B. Lublinsky. Defining SOA as an architectural style. IBM Website, January 2007.

[7] J. Noel. BPM and SOA: Better together. IBM Website, White Paper, 2005.

[8] OMG. Business Process Modeling Notation specification. OMG Website, July 2007.

[9] M. Rosen. BPM and SOA: Where does one end and the other begin? BPTrends, January 2006.

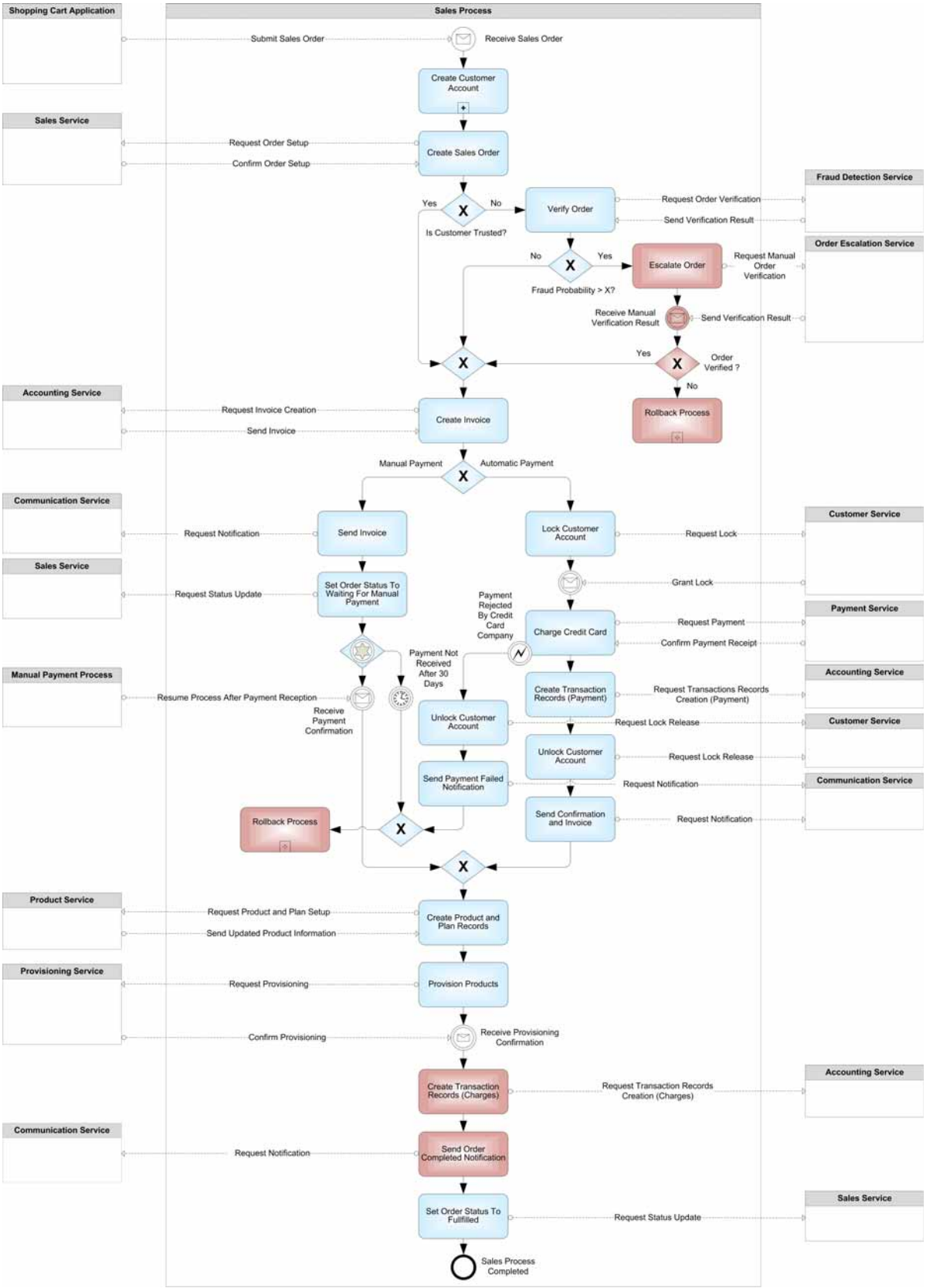


Figure 3. Sales Process BPD

Ontology-Enabled Generation of Embedded Web Services

Klaus Marius Hansen and Weishan Zhang and Goncalo Soares
Department of Computer Science, University of Aarhus
Aabogade 34, 8200 Århus N, Denmark
{klaus.m.hansen,zhangws,afonso}@daimi.au.dk

Abstract

Web services are increasingly adopted as a service provision mechanism in pervasive computing environments. Implementing web services on networked, embedded devices raises a number of challenges, for example efficiency of web services, handling of variability and dependencies of hardware and software platforms, and of device state and context changes. To address these challenges, we developed a web service compiler, Limbo*, in which Web Ontology Language (OWL) ontologies are used to make the Limbo compiler aware of its compilation context, such as targeted hardware and software. At the same time, knowledge on device details, platform dependencies, and resource/power consumption is built into the supporting ontologies, which are used to configure Limbo for generating resource efficient web service code. A state machine ontology is used to generate stub code to facilitate handling of state changes of a device. A number of evaluations show that the design of the Limbo compiler is successful in terms of performance of the generated web service, completeness in being applicable to a variety of embedded devices, and usability for developers in creating new services.*

1 Motivation and introduction

Pervasive computing is becoming a reality. Because of their increasing ubiquity in business environments, web services are increasingly needed to be adopted as service provision mechanisms in pervasive computing environment. Consequently, in a number of applications, web services are deployed on resource-constrained embedded and networked devices. Implementing web services on embedded devices raises a number of challenges. First, embedded devices are constrained in memory, processor and energy resources. The web services should be sufficiently resource efficient in order to provide usable services. Second, development environments for embedded web services must be able to handle the variability of hardware and software, power supply, and possible dependencies between platform properties. At the same time, pervasive computing environments are highly dynamic, with, e.g., device statuses changing very often; something that affects end user applications.

A number of tools and approaches focusing on making web services available on small embedded platforms exist. One example is Microsoft's Web Services on Devices[1], and Fast Infoset[2]. Fast Infoset is not a web service technology per se, but provides a binary encoding of XML that may be used to make web services more efficient in the sense that they use less bandwidth in communication. These tools, however, fall short in the flexibility of code generation and complexity hiding of device details and web service details for the developer. At the same time, they lack the extensibility for using new protocols and technologies, when considering the huge variance of embedded and networked devices.

To address these issues, in this paper, we present *Limbo*, an ontology-enabled compiler for the generation of embedded web services. A number of Web Ontology Language (OWL[3]) ontologies are used to encode device details, platform dependencies, resource/power consumption, and valid Limbo components combinations, which are used to make the Limbo compiler aware its compilation context, such as the appropriate hardware and software for a given service. Runtime states of a device are handled with a state machine ontology and stub code is generated to support reporting device state changes.

The development of Limbo is part of a large, European research project, Hydra[4] that develops secure, service-oriented, and self-managed middleware for pervasive computing application scenarios.

The rest of the paper is structured as follows: in Section 2, we present the design and implementation of Limbo; followed by is the section on how to use the generated code

[1] http://www.microsoft.com/whdc/rally/Rallywsd.mspx
[2] https://fi.dev.java.net/
[3] http://www.w3.org/2004/OWL/
[4] http://www.hydra.eu.com/

for the development of web services. Section 3 discusses ontologies used in Limbo. In Section 4, we present the configuration algorithm used in Limbo. Then we evaluate the Limbo compiler in Section 5, from the perspective of complexity, usability and performance. We compare our work with related work in section 6. Conclusions and future work ends the paper.

2 Limbo design, implementation, and usage

2.1 Limbo design and implementation

Figure 1 shows the module structure of the Limbo compiler. The software architecture of Limbo follows the "Repository" architectural pattern [5] in which a central *Repository* stores data related to the transformation process and on which *Frontends* and *Backends* operate to read and write information. Frontends process source artifacts (in particular web service interface descriptions in the form of WSDL [5] files and ontology descriptions in the form of OWL files). Conversely, Backends produce target artefact's in the form of code (primarily state machine stubs, web service stubs and skeletons) and configuration files.

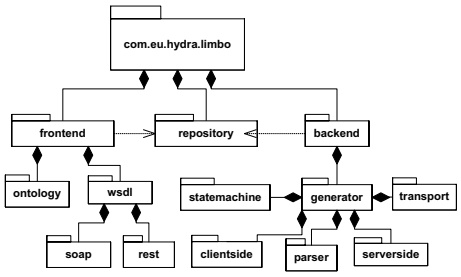


Figure 1. Module structure of Limbo

Backends implement different features. An essential feature is the parser backends with different implementation languages such as Java SE and Java ME (Java Standard Edition/Java Micro Edition). An example of generation can be the generation of client-side stubs and/or server-side skeletons or transport code for network communication between client and a server. To provide the possibility of handling dynamicity of device state changes, a state machine backend generates state machine stubs. Figure 2 shows the compilation process of the Limbo compiler. A "thermometer service" is used to illustrate the compilation and the usage of the generated artifacts. In the example, the service runs on a thermometer device, Pico TH03, and provides a temperature measurement upon request. The following steps are involved:

[5]Web Services Description Language 1.1. `http://www.w3.org/TR/wsdl`

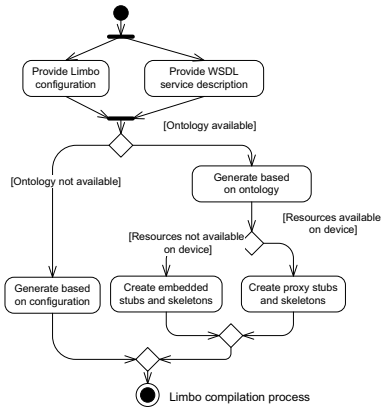


Figure 2. Limbo compiling process

- *Provide WSDL service description*: The main input for Limbo is WSDL file, and Limbo also supports that WSDL files references the Hydra device ontology. An example of a Hydra ontology binding for the thermometer in WSDL would be the following:

```
<hydra:binding device="http://hydra.eu.com/ontology/
                       Device.owl#thermometer"/>
```

  The Limbo ontology front end will resolve this URI and retrieve thermometer hardware and software information.

- *Generation based on configuration or ontology.* If an ontology instance for the device is available, device specific platform information will be used to generate client and/or server code. If the device associated state machine instance available, state machine stub code will be generated. Otherwise, generation configuration is based solely on the developer-supplied parameters.

- *Create embedded/proxy stubs and skeletons.* Stubs and skeletons for the device service are created according to the device's capabilities. If code cannot be directly embedded on the device, proxy code is generated based that will run on OSGi[6]. For the thermometer, as it does not have any computing capability itself, according to the the retrieved platform information from the ontology, proxy code will be generated using OSGi.

2.2 Implementing services based on generated code

For the thermometer with a configuration of a standalone server using Java SE, the following classes are generated:

[6]`http://www.osgi.org`

- *EndPoint.java* - Abstract class that defines the endpoints (i.e. services) that are provided by the server
- *th03OpsImpl.java* - Implementation of the service methods
- *th03Service.java* - A service class that handles requests and returns the respective results
- *LimboServer.java* - Limbo server main class
- *StateMachineStub_Thermometer.java* - State machine stub for thermometer

An OSGi configuration (Java SE) or a Java ME server can be chosen, and Limbo can also generate clients either for Java ME or Java SE. The generation of OSGi code is following the OSGi specification (e.g., an Activator instead of an EndPoint, a servlet instead of a "th03Service"). Classes are also generated to support this in the form of a state machine stub that will allow the service developer to model and notify upon state changes. The following code is generated for the thermometer state machine shown in Figure 3, and the measuring state of the thermometer is linked to the getTemperature service.

```
public class StateMachineStub_Thermometer {
 public void  ThermometerStopping() {  ...  }
 public void  ThermometerStarting(){  ...  }
 public void  ThermometerMeauring(){
  event ev = new event();
  ...
  ev.parts_add(new part("Result",
    "" + service.getTemperature(this.deviceID)));
  eventManager.publish("/statemachine/statechange", ev);
 }
 ...
}
```

Based on the generated artifacts, the device developer needs to implement the device service. This entails:

- *Binding the device services to the actual device.* For the thermometer service this would include, e.g., creating a thread that continuously calculates the temperature and stores the temperature in a local variable. The actual service implementation would then read the value of this variable and return the temperature.
- *Sending state notifications.* The state machine stub needs to be invoked at proper places. In the case of the thermometer, each successive call will at runtime trigger an event being sent through the event manager (Figure 3), when the thermometer is started, when it is measuring, and when it stops as shown in the thermometer state machine in the lower part of Figure 3.
- *Create deployment artifacts.* Next, device and container-specific deployment artifacts (JAR files, OSGi bundles etc.) need to be created in order to be able to deploy the service.

The upper part of Figure 3 shows a typical runtime of a deployed Limbo service. The thermometer service is deployed on a Thermometer Device. A service that needs temperature data ("Thermometer Client") then uses the thermometer service through its web service interface. Thermometer state changes trigger events sent through a publish/subscribe mechanism.

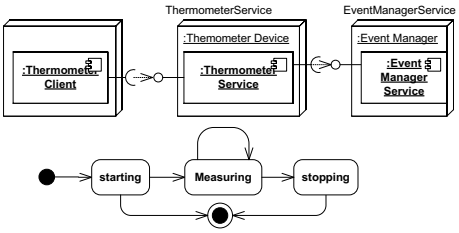


Figure 3. Thermometer runtime and thermometer state machine

3 Ontologies in Limbo

There are a number of reasons for us to use ontologies in Limbo: first, details of device hardware and software, and possible dependencies between them, are hidden in the related ontologies. Web service developers only need to know about the device URI and the service they are implementing, as shown in the Thermometer example. Second, in order to generate resource-efficient code, knowledge on device software platform and resource/power consumption comparisons are built into the related ontologies, and used during the configuration of Limbo for code generation.

We have developed the supporting ontologies for Limbo as shown in Figure 4. The usage of these ontologies can be summarized as follows:

- *LimboConfiguation ontology*. Not all the combinations of the frontends and backends in Limbo are valid. For example, for OSGi, there is no need for the Server generator as a web server is built into OSGi frameworks. Therefore it is very important to regulate the valid combinations of different Limbo components and resolve dependencies among them, whether combinations are explicit in the feature model or implicit. As proposed in [6], we develop a LimboConfiguration ontology to formally specify what the legal feature combinations are.
- *Device ontology and associated hardware platform and software platform ontologies*. These ontologies are used to retrieve device specific information in order to generate resource/power-awareness code for a certain device. The Device ontology is used to define high

level only information of a device, for example device type classification (e.g., an alarm device is a sensor).

The HardwarePlatform ontology includes concepts such as CPU, Memory and so on, and also relationships between them, for example ”hasCPU”. Power consumption concepts and properties for different wireless network are added to the HardwarePlatform ontology to facilitate power-awareness.

The SoftwarePlatform ontology defines VirtualMachine, Middleware and object properties such as *requiresMoreMemory*, *reuqiresFasterCPU*, and their reverse properties. In the Java ontology we define concepts such as JavaVM, JavaByteCode and specify that a specific Java platform (e.g., CLDC) provides a certain Library or Rendering Engine etc.

The OperatingSystem ontology provides a classification of an operating system based on its characteristics and version for example Win32/Win16, which can facilitate the restrictions on which operating system consumes more memory than others.

- *StateMachine ontology*.

  For every type of device in the Device ontology, there is a corresponding state machine instance in the StateMachine ontology. This state machine instance is used to generate state machine stubs.

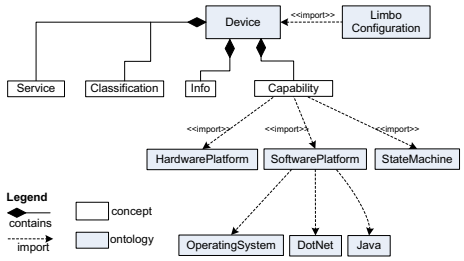


Figure 4. Structure of ontologies used in Limbo

4 Limbo configurations with ontologies

In order to generate resource efficient code, Limbo will utilize the resource/power consumption knowledge built in the ontologies. Therefore the LimboConfiguration ontology imports the Device ontology, and hence all other ontologies through the ontology import mechanism. Object properties in the LimboConfiguration ontology (*requireCPU*, *requireOS*, *requireVM* and *requireLibrary*) are used to specify a backend's detailed requirements for the CPU, operating system, virtual machine, and libraries. The Limbo configuration algorithm is shown as a UML activity diagram in Figure 5 and described next.

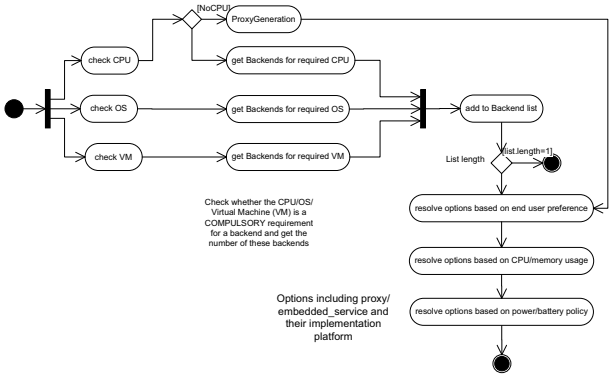


Figure 5. Limbo configuration algorithm

Step 1. Checking CPU/OS/VM details When a compiling task is needed for a certain device, first the detailed software and hardware information, especially CPU, operating system, virtual machine will be retrieved using the ontology frontend.

Step 2. Iteratively checking the backends' required CPU/OS/VM After the detailed information on CPU, operating system and virtual machine has been obtained from related ontologies, this information will be checked iteratively for whether this version of CPU, operating system and virtual machine are required for the backends. This kind of information is stored within instances of the backends associated with the *requireCPU*, *requireOS*, *requireVM* object properties.

Step 3. Resolving choices using user preferences There are situations where we can get multiple options for backends. For example, Motorola MPx220 has Windows Mobile as its operating system, but at the same time it has J2ME MIDP2, which will be compared with end user preferences. Then the generation can go ahead with the chosen platform.

Step 4. Resolving choices based on CPU/Memory usage For situations where memory and CPU usage should be decided, for example J2SE, CDC and CLDC as options, we will choose the one that consumes less memory and requires a slower CPU for small devices as default.

Step 5. Resolve options based on power/energy policy The power consumption of various bearers supported by a device is checked, and choose a corresponding bearer according to the power consumption expectation.

In our implementation of the above algorithm, we are using SWRL[7] to resolve options for multiple platforms as detailed in [2].

5 Evaluation of Limbo

We have evaluated Limbo according to the evaluation framework of one.world [1]. This includes evaluating: *Completeness*: can useful services be generated; *Performance*: is the generated services sufficiently resource efficient; *Complexity and utility*: how hard is it to create services and can others build upon it.

5.1 Completeness

We evaluate this through the generation of services for a set of prototypes for a home automation scenario to testify whether useful services can be generated by Limbo. Here services were primarily created by a member of the Hydra team who has not participated in the development of Limbo (four services) and by a member of the Limbo compiler team (one service). For all services, Hydra helped in hiding web service complexity and in generating efficient web services. The generated services were:

- *Nokia N80 SMS service*. The service uses Limbo's Midlet generation option and runs a Limbo-generated web server.
- *Pico TH03 thermometer service*, *Grundfos Magna 32 pump service* and *Abloy EL582 door lock service*. These services run as proxies on an OSGi gateway and interface with devices via device-specific protocols.

5.2 Performance

Here we report on time and memory usage measurements compared to Apache Axis[8]. The purpose is not to compare to Apache Axis per se since it was designed for a multi-threaded server environment, but rather to see that Limbo-generated services used significantly fewer resources than a popular web service framework.

For measuring resource utilization, we used a setup with a SOAP-based web service implementing an SMS service. This web service was requested by a Limbo-generated client and implemented using Apache Axis and using Limbo on both Java SE and Java ME (on a Nokia N80 mobile phone). For the Apache Axis and Limbo SE implementations a PC (an Apple Mac Book Pro with a 2.33 GHz Intel Core 2 Duo processor, 2 GB DDR2 SDRAM, MAC OS X 10.4.10). The left part of Figure 6 shows the result

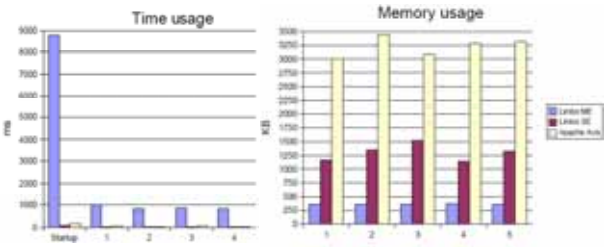


Figure 6. Limbo time and memory usage measurements

of our time measurements with the total execution time for five consecutive calls made to the SMS web service. For all implementations there is a high start-up cost due to the establishment of sockets – in particular so in the Java ME case. The Limbo ME implementation is also orders of magnitudes slower than the SE implementations, a fact that is due to the network setup of the Nokia N80 – and to the fact that the ME implementation actually sends an SMS – since the Limbo SE and Apache Axis implementations are comparable with respect to time usage.

The right part of Figure 6 shows the memory measurements of Limbo and Apache Axis. Both the Limbo SE and the Limbo ME versions use significantly less memory than Apache Axis. In the SE cases, the measurements were made using a JMX agent to measure the maximum amount of memory used during processing of requests. In the ME case, we measured maximum memory with SUN's Wireless Tool Kit (Version 2.5). On average, the Limbo ME service used 362.4 Kb memory. In conclusion, the resource usage of Limbo generated services is significantly smaller than that for Apache Axis-generated services.

5.3 Complexity and utility

Complexity and utility were evaluated by members of the Hydra project that had not participated in the development of Limbo. Two partial evaluations were made on evaluation of ontology construction and code generation. For both, a case of implementing a blood pressure service on an HTCP3300 smartphone[9] was used.

It was possible to create a model of the HTCP3300 device including a state machine within a day of work for an ontology engineer unfamiliar with the device and the associated service. The Limbo compiler has been successfully used to generate small applications to test their compatibility with Windows Mobile Smartphone and Eclipse ME-generated classes.

[7] SWRL homepage. http://www.w3.org/Submission/SWRL/
[8] Apache Axis. http://ws.apache.org/axis
[9] `http://www.europe.htc.com/en/products/htcp3300.html`

5.4 Evaluation conclusions

The Limbo compiler has been shown to be useful with good resource consumption of the generated code. Clearly there is a need for better documentation for both Limbo and the used ontologies, and Windows Mobile concepts of the OperatingSystem ontology need to be improved.

6 Related work

As said in the introduction, existing tools such as Microsoft's Web Services on Devices and Fast Infoset, fall short of the necessary flexibility of generating different code artifacts for the large variant of devices based on different protocols. These tools lack the versatility of being used for different embedded devices.

In Limbo, we translate WSDL files into a local Regular Tree Grammar (RTG) [4] that describes allowed SOAP envelopes as defined by the WSDL files. Though some frameworks can produce grammar-specific parser of XML data such as done by, e.g., XML Screamer [3], our work leverages this work but casts it in the context of web services, where ontologies are used to support the needed configuration during the generation process. The ontologies are helping to achieve generation-context-awareness and help to make decisions on the targeted platform, with the objective of generating resource efficient code.

Apache Muse[10] can simplify the building of web service interfaces for manageable resources. While Muse has a very specialized objective for the targeted specifications, Limbo has a highly flexible architecture which can be easily extended with the generation of code for .NET code, and other specialized platform such as LeJOS[11]. And more importantly is that we are using ontologies and rule languages to rigorously regulate and instruct the compilation, which can bring us some wiser decisions that is not easily achieved by Apache Muse and other existing approaches.

7 Conclusions and future work

There is an increasing requirement to run web services on resource constrained devices in pervasive computing. In this paper, we present an ontology-enabled compiler called Limbo for the generation of embedded web services. Limbo has followed the Repository architecture style where different frontends and backends can be easily added.

Limbo gets device information from the targeted device in compilation from a Device ontology that imports HardwarePlatform ontology and software platform related ontologies, where resource/power consumption comparisons are specified, and used by Limbo to achieve the generation of resource-efficient web services. A StateMachine ontology is used to generate state machine stub code and using an event mechanism to publish the state change events. We are using a LimboConfiguration ontology to rigorously specify the legal feature combination of Limbo compiler.

Our evaluations through the first Hydra prototype show that the design of the Limbo compiler is successful in terms of resource consumption of the generated web services, complexity hiding of the web service itself and that developers can use Limbo to develop resource efficient web services for a variant of different embedded devices.

A more flexible implementation using OSGi is under development. Web service code generation for .Net platform is planed. And more other hardware platform for example LeJOS is also under exploration.

[10] Apache Muse project. http://ws.apache.org/muse/

[11] LeJOS homepage. http://lejos.sourceforge.net/

Acknowledgements

The research reported in this paper has been supported by the Hydra EU project (IST-2005-034891).

References

[1] R. Grimm, D. Wetherall, J. Davis, E. Lemar, A. Macbeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, and S. Gribble. System support for pervasive applications. *ACM Transactions on Computer Systems (TOCS)*, 22(4):421–486, 2004.

[2] K. M. Hansen, G. Soares, and W. Zhang. Embedded service sdk prototype and report. Technical Report D4.2, Hydra Consortium, Dec. 2007. IST 2005-034891.

[3] M. Kostoulas, M. Matsa, and N. e. a. Mendelsohn. XML screamer: an integrated approach to high performance XML parsing, validation and deserialization. *15th international conference on World Wide Web*, pages 93–102, 2006.

[4] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technology (TOIT)*, 5(4):660–704, 2005.

[5] M. Shaw. Some Patterns for Software Architectures. *Pattern Languages of Program Design*, 2:255–269, 1996.

[6] H. Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan. A Semantic Web Approach to Feature Modeling and Verification. In *1st Workshop on Semantic Web Enabled Software Engineering*, Galway, Ireland, Nov 2005. LNCS.

Modeling Services to Construct Service-Oriented Healthcare Architecture for Digital Home-Care Business

Chi-Lu Yang[1,2], Yeim-Kuan Chang[1], Chih-Ping Chu[1]
[1] Department of Computer Science and Information Engineering, National Cheng Kung University
[2] Networks and Multimedia Institute, Institute for Information Industry
[1,2]Tainan, Taiwan R.O.C.
stwin@iii.org.tw, ykchang@mail.ncku.edu.tw, chucp@csie.ncku.edu.tw

Abstract

Using Information and Communication Technologies (ICT) to enable the daily activities and interests such as dining, medicine, lifestyle, traffic, education and entertainment has recently become a world wide trend. Moreover, Service-Oriented Architecture (SOA) is nowadays one of the most important techniques to realize services in industry.

Therefore, we would attempt to give attention to what type of services ICT could realize for chronic patients and how this concept should contribute to their recovery. In this paper, we would like to share our experiences in creating innovative home-care business models. We first discuss the business modeling process, which contains generating care services concepts, investigating market, defining Key Services scenarios and cooperative policies. Second, we present the constructed SOA healthcare platform. Specifically, we explain the technical issues during the development of our business models. Finally, the business models and platform are evaluated using the Key Performance Indicators (KPI) developed in the study.

Keywords: business modeling, SOA design, KPI evaluation, digital home-care

1. Introduction

As information and communication technologies (ICT) has continued to advance, the application of ICT has evolved in the different aspects of our lives such as work and entertainment. In fact, ICT has been gradually infused into our daily lives. Through network techniques, computers could provide many remote services. Interestingly, a large number of innovative service models, such as YouTube, WRETCH, Google, Amazon, etc. are continuously emerging. Using ICT to create innovative services models, many have become famous enterprises worldwide. One of their identical characteristics is that they have applied specific information techniques to integrate network techniques. Also, these models have successfully provided content services to promote business values.

The application of ICT has become a worldwide trend. Infusing Service-Oriented Architecture (SOA) to provide common activities and interests such as dining, medicine, lifestyle, traffic, education and entertainment has also become an objective of industry. Through the SOA platform, we could integrate individual providers into similar service processes. Modeling distinct business providers may also be interesting work. Through this process, researchers would be able to communicate with various stakeholders as well as design systematic platforms.

In this paper, we focus on the domain of patient-centered healthcare in digital home. We then construct the SOA healthcare platform and its application system which provide innovative business models. In the following sections, we will describe the progress of its construction and at the same time share our experiences from this study.

2. Related Works

2.1 SOA principles

Service-Oriented Architecture (SOA) is a software architectural style for realizing and constructing business processes, which are packaged as software services during their life cycle ([1], [2]). SOA defines and reserves IT infrastructure to allow various applications that exchange data in business processes. SOA also separates services into distinct units (components or modules), which can be deployed over the Internet, and can be combined and re-used for business applications. In clearly defined layers of SOA, requirements for business processes could be distinguished and identified. Business requirements are also implemented and combined by distinct software services. Typical layers for SOA are business process layer, business service layer, application integration layer, and technology layer.

Consequently, the general architectural principles [3] point out the ground rules of SOA for its development, maintenance, and use. These are the following:

- Usability - Components or modules would be re-used in various business processes, and even mobile services.
- Compliance to standards - Data exchanges between platforms are important to SOA. These exchanges will extend significant issues for standardization, identification, authorization, etc.
- Service identification and categorization, deployment and delivery, monitoring and tracking, and KPI definition, etc.

One standardizing service could provide diverse and

innovative business processes. A typical example is the service of agential cash-receivers of 7-11 stores in Taiwan which has allowed payments by credit card, telephone, and on-line shopping, to name a few. In addition, the specific architectural principles for design and service definition are categorized into two types. The first type is the interaction between the service consumer and provider. The second type includes the design guidelines of service providers. They are described as:

- Service encapsulation - Various services in the Internet are consolidated with web services under the SOA platform.
- Service loose coupling - Services maintain a relationship that minimizes dependencies on one another.
- Service abstraction - Services are logically hidden from the outside world, beyond what is described in the service contract.
- Service contract - Services attached to the communicable agreements, and defined in service description documents.
- Service reusability – A service is divided into units with extended re-uses.
- Service composition - Collections of units of services can be coordinated and combined to create services.
- Service autonomy – Services have control over the business processes they encapsulated.
- Service optimization – High-quality services are generally considered more than low-quality ones.
- Service discoverability – Services are designed to be accessible to the public, therefore they can be found and assessed via available discovery mechanisms.

Constructing SOA is not only a technical but also a business challenge. In the visions of SOA, relationships between the service consumer and provider are not tightly stipulated. Their relations are loose coupling [4]. Thus, consumer services are not forcefully influenced by the changes made by the providers. Secondly, consumer service interacts with the service provider based on the service contract. Moreover, designing the Service Level Agreement (SLA) is an important task. SLA should also satisfy some general and specific principles.

2.2 CVA Market Demands

Apoplexy, also known as Cerebrovascular Accident (CVA), breaks out when the cerebrovascular suddenly oppilates or fractures. Some patients only experience slightly pathological changes. Others experience paraplegia, and others simply expire. Diseases are distinct based on the position and size of infarcts or hemorrhages. In developed countries [5], CVA is one of the main diseases which result in death or disability. Its occurrence rate is 1.2 - 2.5/1000 [6]. The survival rate is 2/3, but, most patients would suffer disability. Some apoplectics would be discharged from the hospital and taken cared of at home. According to the official statistical data in Taiwan [7], CVA is also included among the top ten causes of diseases. Apoplectics would not only be a difficult illness to manage for the patient, but also to the family and society. Patients with apoplectic diseases need long-term care to reduce pathologies. If they are hospitalized over a long period of time, it will pose financial and emotional burden on their families and will also be a waste of resources in hospitals. Therefore, one effective solution for this situation is to take care of chronic patients at home; this could be a challenge for the family. Because of these perceived needs, the research team poses the following questions:

- What could Information and Communication Technologies (ICT) do for apoplexy patients?
- How should ICT contribute to the apoplectics' recovery?

3. Business Modeling

Business modeling is a critical starting point when we would like to provide care-services on SOA. Before constructing the services platform in SOA, we should ensure our care business models, which include market demands, key services items, services scenarios, market prices, even more cooperative policies, etc. The research process of business modeling for cerebrovascular patients' home-care is described in following figure.

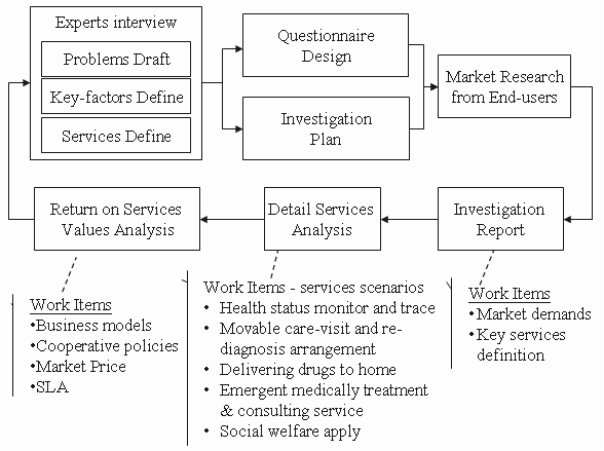


Figure 1. Business Modeling Process

Figure 2. Services Concept

3.1 Services concept

First, researchers iteratively interviewed the domain

experts to draw out the services concept. This was a critical initial step in business modeling. The research efforts were really demanding as about 30% cost of whole project. The services concept is presented in figure 2. Particularly, there are services, roles, proprietors and devices that were included in the services concept.

3.2 Market Investigation

To obtain a good perspective of the market demands, a large-scale market investigation was carried out. The investigation focused on people who lived in Kaohsiung at southern Taiwan. Moreover, the patients' home-care is assumed as the research domain. In addition, the questionnaire was designed based on the services concept. Consequently, the investigation was divided into two parts namely the quantitative investigation and qualitative investigation, and were performed concurrently. Details of the investigation were recorded in the technical report [8]. Quantitative Investigation Results were significantly listed in table 1.

Table 1. Quantitative Investigation Results

Items	Demands of home-care	Percentage
1	health status inspection (Notify / Arrange / Trace)	61.2%
2	Health status monitoring, tracing and unusual alerting by medical equipments	56.2%
3	Assisting or accompanying to take medical treatment (register at a hospital / ambulance)	23.1%
4	Emergency medical treatment and notifying family members	48.5%
5	Consulting Medical treatment	47.7%
6	Providing supplementary instruments for home-care	49.2%
7	Serving routines at home	21.5%
8	Nutrition consulting	26.9%
9	Psychological consulting	26.2%
10	Assisting to call an ambulance	20.8%
11	Assisting to apply for social services	53.1%
12	Entertainment activities	21.5%

The results of the qualitative investigation are as follows:

1. For the cerebrovascular patients, restoring limbs at home has positive effects as they enhance the patients' movability.
2. The majority of the caregivers are female. When family members work during day time, caregivers who majorly come from overseas take care of the patients.
3. The major economic resources of the elders are their children as they depend on their previous savings.
4. Members of the family trend to drive their patients from their homes to the hospitals by themselves. However, when special equipment is needed, they choose to access the transportation services provided by the care centers.
5. Persons who have work hope that elders' day to day needs could be taken cared of by the hospitals or governments which are supported by community volunteers. Workers hope that care center could directly provide them more of the information about their volunteers.
6. There should be a higher provision of equipment to the patients who need them more, especially the equipment that regularly monitors the patients' status and alerts emergency situations. This is considered an innovative concept. Through the equipment, someone could efficiently take care of patients whose statuses were unusual.
7. Hospitals should charge a minimal fee for these types of services. The families would like to pay the services within the limit of their income. The maximum amount they would like to pay is 5,000 NT dollars per month.

3.3 Define Key Services to Develop

We would like to identify important services that should be developed from this investigation report. Based from our market analysis, we could first fund six items whose percentages are more than 40%. Their item numbers are 1, 2, 4, 5, 6 and 11. The other items, whose percentages are less than 40%, are filtered out. Second, we would only select item 1, 2, 4, 5 and 6 into key services group. The reasons are listed in last two columns of table 2. We filtered out item 11 since it could be supported without information techniques. Key service items in the group would be analyzed to form services scenarios for future development. Services scenarios were recorded into document [9].

3.4 Cooperative Policies and Market Prices

Home-care services are considered comprehensive solutions for patients at home because single service provider could not successfully provide these services. In addition, service providers should be organized into a virtual organization as they serve as care centers in the community, hospitals, IT companies, transporters, and insurance companies. Partnerships and agreements among these stakeholders have to be documented through contracts. Through this strategy, they could merge the services supply chains. Specifically, the insurance companies play a special role in the business models. During the services life cycle, service providers should review insurance policies, since there might be some risks in caring patients at home.

After planning the cooperative policies, we should also define the market prices for each service item. Care centers could sell services to end-users and enter into contracts with them based on the market prices of the combined services. In addition, market prices would be defined based on the previous investigation and the cost of different service scenarios. Thus, the cost of each service should be calculated. We listed the factors of each service's cost in the table 3.

Table 2. Select Services Items into Key Services Group

Discussion / Key Services Group	Investigation Results	Baseline to develop	Techniques / Business issues
Freelance care supervisors and re-diagnosis arrangement	health status inspection (Notify / Arrange / Trace) **61.2%**	In order to save medical resources for severe patients, doctor would suggest patients, who do not need much medical attention, to go to local clinics. This means that patients who only experience slight discomfort should just go to the clinics for medical. This system will make the distribution of patients in the hospitals even	To exchange diagnoses records
	Consulting Medical treatment **47.7%**		
Delivery of medicines to the patient's home	Providing supplementary instruments for home-care **49.2%**	By providing this innovative service, hospitals would have an increase in extra income. Patients could also get their medication more conveniently.	To integrate distinct industries
Health status monitor and trace	Health status monitoring, tracing and unusual alerting through modern medical facilities **56.2%**	Collecting bio-signals from patients at home and monitoring their variations are critical techniques. Hospitals could precisely get the patients' conditions.	To collect and to store bio-signals
	Emergency medical treatment and notification of the patients' condition to family members **48.5%**		

Table 3. Factors of Service Cost for Business Models

Factors	Price Unit
Expected number of centre-carer visiting	person-time/month
Human resources and qualifications	Person Quantity
Budget for human resources	dollars/year
Budget for IT systems and devices	dollars/month
Transporter's fee	dollars/month
Total budget	Dollars/year
Fees to be charged against user for their remaining balance	dollars/one person a month

4. Architecture Constructing

Developing SOA services platform and its application is another important task in providing SOA business models. To support this initiative, we employed and adopted software lifecycles [10] to develop them. Development process and its work items are described in figure 3. In the high-level design phase, we designed the SOA healthcare platform based on the services demands. An application system was analyzed through knowledge engineering in the requirement phase, and was designed using the MVC methodology in detail design phase. The required functions of these application systems were divided into three parts - user interfaces, business logics, and data models. User interfaces would be implemented in the application level while business logics and data models would be supported by SOA services platform. More detailed functions for each module are categorized into functional, non-functional, interface, and security. The high-level modules of the application system are showed in figure 4.

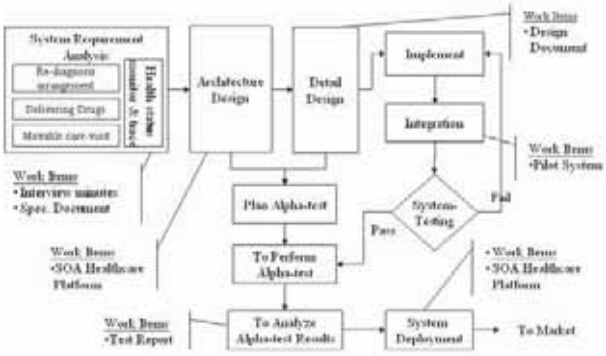


Figure 3. Developing Process of SOA Healthcare Platform

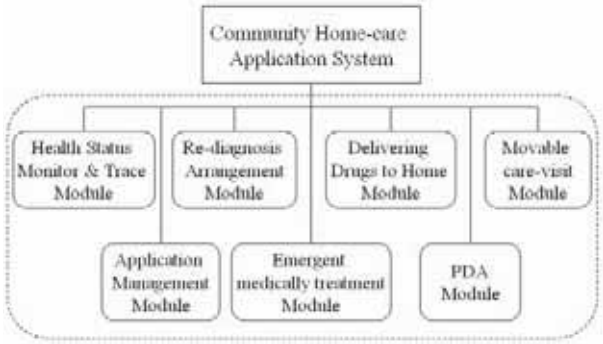


Figure 4. Functional Architecture of Application

4.1 SOA healthcare Platform

SOA healthcare platform was designed to provide executable environments which support standardized messages, various interfaces and flexible connections.

Different services techniques could cooperate with each others on this platform. In this platform, each module and component should be developed under MVC methodology. Business logics and data presentations should be separated into different independent components. Developing business services should focus on the design of business logics. The other technical components could be simply dealt through the SOA platform. SOA Healthcare Platform is showed in figure 5. Services-flow control tool contains three main modules, specifically the service executing engine, service process defining and services monitoring. The module of service process defining is used to identify service processes of application system. Moreover, the module of services monitoring is used to check the current statuses of service objects in the application system. Finally, the module of service executing engine is used to bind service objects and the other modules in platform. There are three healthcare tools that are used by patients in the platform. These include bio-signals management, messages management and end-users management. The module of bio-signals management is used to supervise patient's bio-signals from the homebox, an end-device installed in the patient's house. Meanwhile, the module of end-users management is used to handle users' profiles and personal descriptions. The module of messages management is used to handle messages passing between internal and external objects. In addition, the logger module is used to record the histories of the events while the module of exception handler is used for tracing run-time defects in platform. We would like to call these modules' components based on the coding style below:

```
try {
    // call regular components…
} catch {
    // call exception handler's components …
} finally {
     // call logger's components…
}
```

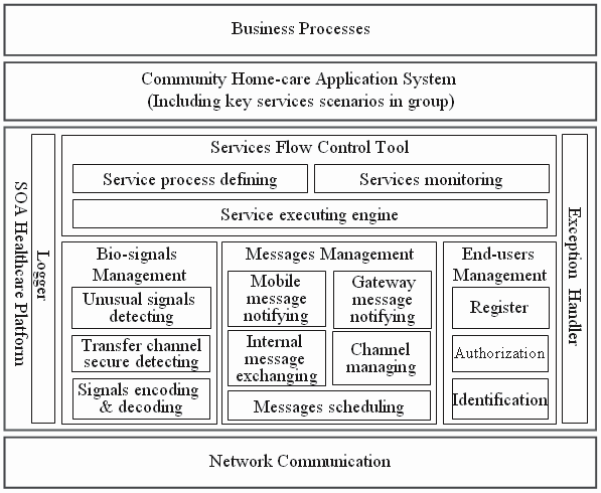


Figure 5. SOA Healthcare Platform

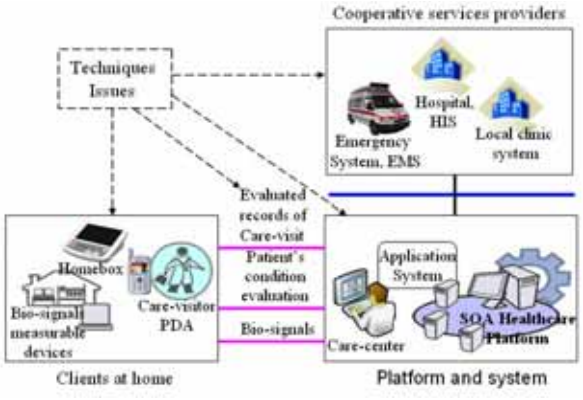


Figure 6. Deployment Overview of Systems

4.2 Surmounting Techniques Issues

We plan to deploy our systems in a real environment based on the services concept. The SOA healthcare platform was deployed to administer the services in the care center side and manage the homebox, bio-signal devices installed in the clients' house. The care center supervisor would regularly bring his/her PDA and visit the patients at their homes. Some cooperative services providers, such as hospitals HIS, local clinic system and emergency system, are also showed in figure 6. Since this is considered as a home-care solution really deployed in three sides, some technical issues about home-care will occur. By then, we will have to address these issues using the SOA platform. Furthermore, technical issues in home-care are addressed and shortly described as follows:

- Transferring bio-signals from clients to the application server in real-time
  - Transfer channels are always maintained by the platform which supports the message exchange.
  - Unusual bio-signals are monitored and handled by the platform through the application server.
  - End-users' profiles are managed by the platform. Private profiles should be shared in a secure way.
  - Messages and bio-signals should be standardized between systems through the platform.
- Integrating multiple bio-signals devices
  - Homebox would integrate multiple bio-signal devices. The condition of the homebox should be regularly checked by the platform, since it was registered there.
  - Within the homebox, user interfaces will simply be implemented to interact remotely. The interactions are executed through the platform.
  - If abnormal network occurs, homebox could detect and reconstruct by itself. After its reconnection, the homebox should pass exceptional logs to the platform.
- Integrating homogeneous providers
  - Business services processes should be composed and executed in an efficient way through the platform. The status of the executing services should also be monitored through the platform.
  - Caregivers should maintain terminal systems, which directly connect them to the platform.

Diagnostic records should be carefully shared using a standardized format between providers.

- Homogeneous systems should be integrated through the platform using for healthcare area.

Detailed specifications of this design are documented in technical report [11].

5. Key Performance Indicators Evaluation

The business models are evaluated using the Key Performance Indicators (KPI). We define our KPI by referring to the Balanced Scorecard (BSC) models [12]. For the SOA healthcare platform and business models, we planned to have four presentations of KPI. They are categorized as system quality, customers' satisfaction, business achievements and financial achievements. The measurements are described in the following.

1. System quality would be measured and analyzed according to its user-friendliness, level of security and privacy during the data transfer, the defects rates, the services response time and the used times.

2. Customers' satisfaction would be measured and analyzed according to the following criteria: marketing share, customers' continuity, increase in the number of customers, increase in customers' satisfaction, and stakeholders' satisfaction.

3. Business achievements would be measured and evaluated according to business quantity, number of customers, number of, transactions and profit growth.

4. Financial achievements would be measured and evaluated based on the profit growth and service packages, management performance and financial forecast that are based on the three phases, which are establishing phase, increasing phase and mature phase.

6. Conclusions

As information and communication technology (ICT) advances, the applications of ICT should provide convenience and business solutions to people. As revealed in our study, Service-Oriented Architecture (SOA) platforms could smoothly integrate business models and combine distinct services providers for innovative services. In this paper, we shared our experiences on creating innovative home-care business models. We presented how we created the SOA healthcare platform and addressed technical issues that emerged during its development. Finally, we demonstrated how we plan to evaluate the business models and platform using the four presentations of KPI. We believe that this is an interesting concept and a good case study on modeling services for the Service-Oriented Architecture in business processes.

7. Acknowledgements

This research was supported by the Applied Information Services Development and Integration project of the Institute for Information Industry (III) and sponsored by MOEA, Taiwan R.O.C.

Interviewed care-experts and investigated patients were supported by the Department of Medical Information, Chung-Ho Memorial Hospital in Kaohsing, Taiwan R.O.C.

8. References

[1] Thomas Erl, "Service-oriented Architecture: Concepts, Technology, and Design," Prentice Hall PTR., July, 2005.

[2] Dave Hornford, "Definition of SOA," The Open Group, October, 2006.

[3] Yvonne Balzer, "Improve your SOA project plans," IBM Global Services, July 2004.

[4] Eric Newcomer and Greg Lomow, "Understanding SOA with Web Services," Addison Wesley, January, 2005.

[5] Liesbeth Bergman, MD, Jan H.P. van der Meulen, MD, Martien Limburg, MD, J. Dik F. Habbema, "Costs of Medical Care After First-Ever Stroke in the Netherlands, Stroke, 26, pp1830-1836., 1995.

[6] Pullen R, Harlacher R, Pientka L, Fusgen I, "The elderly stroke patient - observations 18 months after the event," Z Gerontol Geriatr, 32(5), pp358-363, 1999.

[7] Department of health, executive Yuan, Taiwan R.O.C., "Death causes statistical information in 2006 year," http://www.doh.gov.tw/statistic/index.htm, 2006.

[8] ISAT Team, "Investigation Report about Analyzing Home-care Services Demands of Apoplectics," Institute for Information Industry, Taiwan (R.O.C.), November, 2005.

[9] ISAT Team, "Evaluation Report about Planning and Establishing Home-care Business models," Institute for Information Industry, Taiwan (R.O.C.), December, 2005.

[10] Roger S. Pressman, "Software Engineering: A Practitioner's Approach," Mc Graw Hill, sixth edition, 2005.

[11] ISAT Team, "Design Report about Home-care System Architecture and Functional Specifications," Institute for Information Industry, Taiwan (R.O.C.), January, 2006.

[12] Kaplan R S and Norton D P, "Balanced Scorecard: Translating Strategy into Action," Harvard Business School Press, 1996.

[13] Mohammad Abu-Matar and A. Jefferson Offutt, "Service Oriented Architecture Empirical Study," the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2007.

[14] Kuo-Wei Hwang, "Information Services in Service Oriented Architecture-Challenges and Opportunities," the 18th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2006.

Testing Relational Database Schemas with Alternative Instance Analysis

Maria Claudia F. P. Emer
State University of Campinas
mcemer@dca.fee.unicamp.br

Silvia Regina Vergilio
Federal University of Paraná
silvia@inf.ufpr.br

Mario Jino
State University of Campinas
jino@dca.fee.unicamp.br

Abstract

Databases are employed to store a great amount of data extremely important for business operations. Database test evaluates if a database meets its requirements. In this context, to test the database schemas is a fundamental activity to increase the confidence on the integrity of the data being manipulated by the database applications. There are few works that address this subject. Approaches to test database schemas can help to find faults related to the incorrect or absent definitions of constraints in the data and can contribute to avoid failures in the applications. With this in mind, we present a fault-based testing approach for database schemas and introduce testing criteria based on the classification of the most common types of fault in database schemas. In our approach database instances and queries are used to test the schemas. The instances are generated according to patterns defined for fault classes and represent possible faults. Preliminary results are discussed.

1. Introduction

Testing activity in software development contributes to generate reliable products and to evaluate software quality. Test techniques and criteria have been proposed to guide the test process, which may involve many correctness issues in software applications. For instance, database testing aims to evaluate how well a database meets its requirements and can determine query response time, data integrity, data validity and data recovery [10]. Database system testing involves many correctness aspects, such as [4]: application behaviour with respect to the specification; how well the database schema models the real world; accuracy of data in the database; safety and privacy; and correct execution of insertions, updates and deletions of data and information from data schemas.

Schemas are frequently used in data base applications to define the logical structure and the relationships among data. Schemas are designed according to the data specification. Testing of database schemas can contribute to increase the confidence in the integrity and accuracy of the data being manipulated. If incorrect data is validated by a schema and passed to the application, this may cause a failure. In spite of its importance, the testing of schemas has not been a popular subject in the area. Most of the works address data generation, application and database design testing [2, 3, 4, 5, 7, 11, 14, 16, 17].

In a previous work [9] we introduced an approach to test schemas which considers generic fault classes and which can be applied to any schema that can be represented by a model based on a MOF specification [13]. We have explored its use mainly in the context of XML [8]. The promising results motivated us to investigate the use of such approach in the context of relational database schemas [15], a largely used kind of schema.

To do this, we instantiated the generic fault classes, proposed in [9], for the context of relational database and of the entity-relationship (ER) model [6], a very popular model employed in database applications. By considering the approach and the presented fault classes, we introduced a set of testing criteria to be used for evaluating the testing activity. We implemented a support tool and conducted experiments to evaluate the criteria in this new context.

The remainder of this paper is organized as follows. Section 2 presents the fault-based testing approach adapted to the relational database context. This section introduces the fault classes and the testing criteria. Section 3 presents the experimental results. Section 4 describes related work. Section 5 contains the conclusions and directions for future work.

2. Alternative Data Instance Analysis

In this paper, the proposed fault-based testing approach is named Alternative Data Instance Analysis (ADIA) and is adapted to the context of relational database schemas. It has the goal of revealing constraint faults related to the definition of entities, attributes, relationships and semantics in a database schema. These faults can be related to issues such as incorrect or absent restriction definitions for those schema elements. The idea is to evaluate these issues to avoid faults in the database schema that can affect data integrity in the database and cause failures in the database application.

ADIA is fault-based and includes database instance alternatives and queries to reveal the faults. The data models that represent schemas, the fault classes identified in the database schema, the schema representation and the test process are presented next.

2.1. Data Model

The data model is represented by a metamodel *M* defined according to the MOF (Meta-Object Facility) Specification [13]. Figure 1 illustrates the metamodel *M*, described in UML notation [1], consisting of the following classes: Element (elements or entities); Attribute (properties of the elements) and Constraint (restrictions associated to elements and attributes).

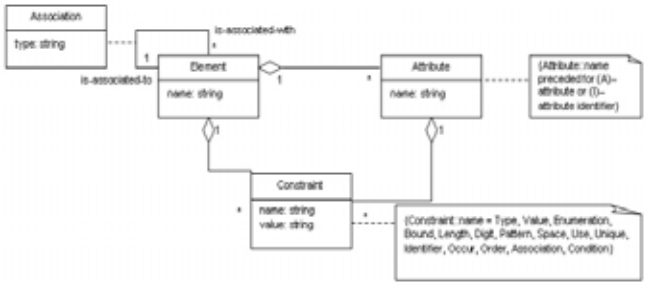


Figure 1. Metamodel *M*

To illustrate the data model in this context, consider a fragment of an ER diagram (Figure 2). The diagram describes data on students from a university: course and type of course. Figure 3 shows the correspondent class diagram based on *M*.

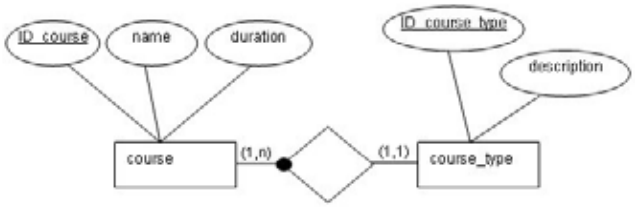


Figure 2. Fragment of an ER diagram

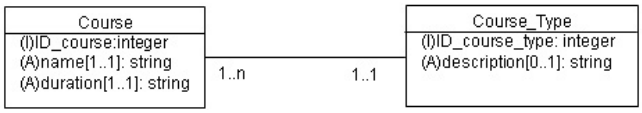


Figure 3. Data Schema for Course Data concerning the Metamodel *M*

2.2. Fault Classes

ADIA is fault-based; hence, common faults introduced during the conceptual design of a schema are organized into four groups of constraints: domain, definition, relationship and semantic. These faults are identified through analyses of data schemas. In this paper, the fault classes presented previously were instanced for relational database schemas. Table 1 presents the fault classes concerning relational database schemas.

Table 1. Fault classes for relational database schemas

Fault Class	Description
Group 1 (G1) - Domain Constraints	**faults related to domain definition of attribute values**
IDT - Incorrect Data Type	incorrect definition of data type
IV - Incorrect Value	incorrect definition of default value
IEV - Incorrect Enumerated Value	incorrect definition of the list of acceptable values
IMMV - Incorrect Maximum and Minimum Values	incorrect definition of upper and lower bound values
IL - Incorrect Length	incorrect definition of number of characters allowed for values
ID - Incorrect Digits	incorrect definition of total amount of digits for numeric values
IWSC – Incorrect White Space Characters	incorrect definition of how white space characters must be treated
Group 2 (G2) - Definition constraints	**faults related to data integrity**
IU - Incorrect Use	the attribute is defined incorrectly as optional or obligatory
IN – Incorrect Uniqueness	the attribute is defined incorrectly as unique
IK - Incorrect Key	the attribute is defined incorrectly as primary key or foreign key
Group 3 (G3) - Relationship Constraints	**faults related to relationship definition among entities**
IO - Incorrect Occurrence	incorrect definition of number of times a same entity may occur
IC - Incorrect Association	incorrect definition of an association: cardinality, generalization/specialization, aggregation, associative element
Group 4 (G4) - Semantic Constraints	**faults related to constraints definition in relation to data content expressed by business rules**
IO - Incorrect Condition	incorrect definition of predicate expressed for a condition that must be satisfied by attributes

2.3. Formal Representation

A formal representation is used to process data schemas, by providing the identification of the entities, attributes, constraints and of the associations among them. A data schema S is denoted by $S = (E, A, R, P)$, where:

- E is a finite set of entities;
- A is a finite set of attributes;
- R is a finite set of constraints concerning domain, definition, relationship and semantics associated to the elements and attributes;
- P is a finite set of association rules among elements, attributes and constraints. Consider $U = E \cup A$. The association rules are represented by:
  - $p(x, y) \mid x, y \in E \wedge x \neq y$;
  - $p(x, r) \mid x \in E \wedge r \in R$;

- $p(x, r, SU) \mid x \in U \wedge r \in R \wedge SU = \{u_1, u_2, ..., u_m\} \subset U$, $\forall u_i \neq x,\ 1 \leq i \leq m, m \geq 1$, where m is the number of elements and attributes in SU.

Example 1 shows the notation used to describe a data schema.

Example 1: Formal representation for ER diagram of Fig. 2

$S = (E, A, R, P)$

$E = \{course, course\_type\}$

$A = \{ID\_course, name, duration, ID\_course\_type, description\}$

$R = \{type,\ key, use, length, uniqueness, association\}$

$P = \{p_1(course, ID\_course), p_2(course, name), p_3(course, duration),$
$p_4(course, association, course\_type),$
$p_5(course, key, course\_type),$
$p_6(ID\_course, type), p_7(ID\_course, key), p_8(name, type),$
$p_9(name, use), p_{10}(name,\ length), p_{11}(duration, type),$
$p_{12}(duration, use), p_{13}(course\_type, ID\_course\_type),$
$p_{14}(course\_type,\ description),$
$p_{15}(course\_type,\ associative, course),$
$p_{16}(ID\_course\_type, type), p_{17}(ID\_course\_type, key),$
$p_{18}(description, type), p_{19}(description, use),$
$p_{20}(description, length), p_{21}(description, uniqueness)\}$

2.4. Testing Process

The schema under test and the corresponding database instance (original database instance) are provided by the tester. These entries to the testing process are presented in Examples 2 and 3. Example 2 shows a fragment of the schema written in the DDL (Data Definition Language) script related to the ER diagram of Figure 2. Example 3 presents a sample of the original database instance associated to the schema of Example 1.

Example 2. DDL related to the ER diagram of Fig 2.

```
CREATE TABLE course (
        id_course           int IDENTITY,
        name                varchar(40) NOT NULL,
        id_course_type      int NOT NULL,
        duration            int NOT NULL
)
go
ALTER TABLE course
        ADD PRIMARY KEY NONCLUSTERED (id_course)
go
```

Example 3. A sample of the original database instance

Course			
Id_course	**Name**	**id_course_type**	**duration**
1	Computer science	1	4
2	Computation Engineering	1	5
3	Database	3	2
4	Images Processing	4	2

Initially, the representation S of the schema under test is built (Example 1). Based on S, schema elements (entities, attributes, relationships among entities and semantic constraints related to attributes) are associated to the fault classes. These associations are named fault associations. Table 2 presents some associations identified in S for its entities and attributes.

Table 2. Fault associations identified in S

Entity/Attribute	Fault Class
Course	Incorrect Association (cardinality)
	Incorrect Key
ID_course	Incorrect Data Type
name	Incorrect Use
	Incorrect Data Type
	Incorrect Length

Additionally, the tester can identify other fault associations among schema elements and fault classes. This provides the detection of faults that could not be detected with the fault associations identified through representation S. Thus, the tester can determine fault associations to reveal faults related to absent constraint definitions. These are faults covered by the fault classes.

Next, these fault associations are selected. The selected fault associations are used to guide the generation of the alternative database instances, indicating the schema elements that should be modified in the original database instance and the fault classes which define the modification patterns that should be applied.

The alternative database instances are generated through modifications in the original database instance. These single modifications are made by insertions and changes into records of the original database instance according to patterns defined for each fault class. These modifications are representative and sufficient to detect the fault of each fault class. For example, to reveal a fault related to G1-IL (Group 1 - Incorrect Length) in the attribute $a \in A$ of the database schema S, a record should be altered, for example, with a number x of characters out of the bound allowed for attribute a in an alternative database instance generated. Example 4 illustrates an alternative database instance for table Course of the original database instance presented in Example 3. In Example 4, the attribute name is associated to fault class G1-IL (Group 1 - Incorrect Length).

The records of the generated alternative database instances are separated into valid or invalid with respect to the schema under test; that is, a record generated by modification patterns may not be in conformity with the schema under test. An invalid record is not accepted in the alternative instance; thus, this alternative instance is not queried, but it is part of the test result.

Example 4. Alternative database instance

Course			
Id_course	**Name**	**id_course_type**	**Duration**
1	Computer science xxxxxxxxxxxxxxx xxxxxxxxxxxxx	1	4
2	Computation Engineering	1	5
3	Database	3	2
4	Images Processing	4	2

The selected fault associations also guide the generation of the queries. The queries are automatically generated as SQL statements according to the query patterns associated to each fault class which can be detected in the schema. Queries for each selected fault association are generated and executed for each correspondent valid alternative database instance.

A test data is formed by a valid alternative database instance and a query to this alternative instance. Expected results for the test data are obtained from the database specification. The tester compares test results with the specification to discover faults in the database schema.

2.5. Testing Criteria

The testing criteria based on the fault classes aim to guide the testing process by selecting fault associations to be exercised and, consequently, by reducing the application costs of the ADIA through the selection of a subset of fault associations. The costs are related to the number of alternative database instances and queries generated. However, the effectiveness of the ADIA to reveal data schema faults should not be reduced.

In that way, the testing criteria are used to select fault associations to be exercised in the testing process. These criteria are based on the fault classes for relational database schemas. They request that the fault associations are exercised through query execution on the alternative data instances related to these associations. Consider z an element or attribute. These criteria are:

All constraints – all fault associations with regard to z related to fault classes of the groups of domain, definition, relationship, and semantic constraints must be exercised;

All domain constraints - all fault associations with regard to z related to fault classes of the group of domain constraints must be exercised;

All definition constraints - all fault associations with regard to z related to fault classes of the group of definition constraints must be exercised;

All relationship constraints - all fault associations with regard to z related to fault classes of the group of relationship constraints must be exercised;

All semantic constraints - all fault associations with regard to z related to fault classes of the group of semantic constraints must be exercised;

All constraints groups – at least one fault association with regard to z related to each group of domain, definition, relationship and semantics constraints must be exercised, if such association exists.

3. Case Study

Our case study uses a database application developed by graduate students, containing data on university students: personal, academic and professional data. Testing process was performed during the development of the database. The ER schema for the application has 19 entities and 20 relationships. Relational database application was implemented using PostGreSQL.

The testing process was performed using XTool [12], a tool that supports the testing approach for database schemas described in the Section 2.4. XTool was developed in Java and tests relational database schemas by using JDBC (Java Database Connectivity) to manipulate and query the information in PostGreSQL database.

XTool found 19 entities and 73 attributes in *S*. These entities and attributes were automatically associated to fault classes in the schema under test. Moreover, the tester found other fault associations among schema elements and fault classes. By using the selected fault associations, XTool generates alternative database instances and SQL queries automatically. The total number of fault associations (identified by XTool and the tester) was 297 and 1,240 queries were executed.

We used all the testing criteria to select the fault associations, except the "all constraints groups" criterion. The idea is to compare the use of the testing criteria by analyzing costs and revealed faults in this database schema.

Table 3 presents an example of fault associations, the number of records modified in the original database instance to generate the alternative instances and the number of generated queries for the entity "Academic_records" and some of its attributes according to the testing criteria.

The tester had the task of comparing the queries results with the expected ones. Records of invalid alternative database instances generated were also considered test results and used during the testing analysis. Table 4 shows the number of generated queries and valid alternative instances by faults revealed for the testing criteria applied by using XTool.

The faults revealed in the test process are related to: the incorrect definition of data type and length constraints; the absence of constraints and enumerated values for attributes; and, incorrect cardinality for a relationship. Faults of absence of constraints were revealed with the fault associations found by the tester.

The testing criteria that revealed faults are: "all domains constraints", "all definition constraints" and "all relationship constraints". Hence, when we applied the testing criterion "all constraints", all faults detected with the other criteria were also revealed. In this case, we can see that the number of queries that revealed faults was 14% of the total number of queries generated and the number of

valid alternative instances that represented revealed faults was 15% of the total number of valid alternative instances generated. Those results are an indication that the testing criteria can help to reduce the application costs of ADIA.

Table 3. Fault associations found in S, number of modified records and generated queries for entity Academic_records

Testing Criterion	Fault Associations		Number of records	Number of queries
	Entity/ Attribute	Fault Class		
All Relationship Constraints	Academic_ records	Incorrect Association (cardinality)	41	12
		Incorrect Association (Associative Element)	6	1
All Definition Constraints	Academic_ records	Incorrect Key	1	1
		Incorrect Uniqueness	1	1
All Domain Constraints	Begin_date	Incorrect Data Type	7	1
All Definition Constraints	Begin_date	Incorrect Use	4	1
All Semantic Constraints	Begin_date	Incorrect Condition	48	25

Table 4. Faults revealed

Testing criterion	Fault classes	Number of queries	Number of valid alternatives	Number of faults revealed
All Domain Constraints	Incorrect Data Type	18	13	5
	Incorrect Length	15	12	3
	Incorrect Digits	44	40	4
	Incorrect Enumerated Value	5	3	1
Total by criterion		**82**	**68**	**13**
All Definition Constraints	Incorrect Use	85	72	13
Total by criterion		**85**	**72**	**13**
All Relationship Constraints	Incorrect Association (cardinality)	7	6	1
Total by criterion		**7**	**6**	**1**
All Constraints	Incorrect Data Type, Length, Digits, Enumerated Value, Use, Association (cardinality)	174	146	27
Total by criterion		**174**	**146**	**27**

Revealed faults were removed and the use of ADIA contributed to improve the quality of the tested application.

It is important to remark that the original database instances are not replicated by XTool to generate the alternative instances. The original instance is updated with a single modification pattern, queried by the generated query and the modification is undone.

4. Related Work

As mentioned previously, many works in the literature address the testing of database applications [2, 4, 5, 7, 11, 16, 17]. Some of them [3, 14] propose the use of schema information to test the application. Robbert and Maryanski [14] use information obtained from the database schema to generate a test plan for the database application, indicating points that should be verified in the test. Chan et al. [3] propose a fault-based approach to test SQL statements of database applications using information captured from the conceptual data model to generate SQL statement mutants. They use schema information to test the application or SQL statements, but they do not address schema testing.

The abovementioned works do not have the goal of validating schemas, the focus of the present paper. Our approach has a different objective and contributes to increase the reliability of the data stored in the database.

5. Conclusions

The fault-based approach ADIA for database schemas and the testing criteria were derived from our previous work [8, 9]. The main goal of this testing approach is to reveal faults in the database schema to ensure the quality of the data stored in the database and, consequently, to contribute to increase the reliability of the database application. Data integrity in databases is fundamental to avoid incorrect data processing resulting in failures in the database application.

ADIA contributes by: presenting a metamodel and a formal representation for database schemas; defining fault classes for database schemas based on the entity-relationship model; introducing testing criteria based on fault associations, generating alternative database instances based on fault classes; and using queries to reveal faults.

XTool was implemented to support ADIA and it does not need the database application to test the database schema. XTool needs only the schema to be tested and the original database instance. In addition to that, the alternative database instances generated automatically by XTool could be used to test the applications that access the database.

The case study performed by using XTool shows that ADIA is effective in revealing faults covered by the fault classes in database schema and that the largest cost is related to the analysis of the results by the tester; the tester compares the test results with the expected ones from data specification. In addition to that, the results indicate that the

testing criteria can help to reduce the application costs of the ADIA.

ADIA was used to reveal faults in schemas of relational database model. It can also be used to detect faults in schemas of other database models, for instance, XML databases.

Other experiments are necessary to better evaluate the effectiveness of the testing approach and criteria for detection of faults in the context of database models. We intend to use the testing approach to reveal faults in more complex integrity constraints or faults involving several relations.

In addition to that, in future work we intend to analyze the relationship between the number of faults/fault classes and instances/queries that can be redundant; furthermore, a way to reduce the number of instances/queries by using test criteria should be proposed and evaluated empirically.

6. Acknowledgments

We acknowledge the partial financial support from the Brazilian Research Agency (CNPq).

References

[1] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[2] CHAN, M.; CHEUNG, S.. *Testing Database Applications with SQL Semantics*. In Proc. of the 2nd Intl. Symp. on Cooperative Database Systems for Advanced Applications, pp 364-375, March 1999.

[3] CHAN, W. K.; CHEUNG, S.C.; TSE, T. H.. *Fault-Based Testing of Database Application Programs with Conceptual Data Model*. In Proc. of the 5th Intl. Conference on Quality Software, pp 187-196, 2005.

[4] CHAYS, David; DAN, Saikat; FRANKL, Phyllis G.; VOKOLOS, Filippos I.; WEYUKER, Elaine J.. *A Framework for Testing Database Applications*. In Proc. of the 2000 ACM SIGSOFT Intl. Symp. on Software Testing and Analysis, Vol. 25 Issue 5, August 2000.

[5] CHAYS, D.; DENG, Y. *Demonstration of AGENDA Tool Set for Testing Relational Database Applications*. In Proc. of the 25th Intl. Software Engineering Conference, 2003. IEEE Computer Society, pp 802 – 803, May 2003.

[6] CHEN, P. P.. *The Entity-Relationship Model – Toward a Unified View of Data*. ACM Transactions on Database Systems, Vol. 1, Nº 1, pp 9-36, 1976.

[7] DENG, Yuetang; FRANKL, Phyllis; CHAYS, David. *Testing Database Transactions with AGENDA*. In Proc. of the 27th Intl. Conference on Software engineering. ACM Press, May 2005.

[8] EMER, M.C.F.P.; VERGILIO, S.R.; JINO, M.. *A Testing Approach for XML Schemas.* In Proc. of the 29th Annual Intl. Computer Software and Applications Conference, Vol. 2, pp 57 – 62, July 2005.

[9] EMER, M.C.F.P.; VERGILIO, S.R.; JINO, M.. *Fault-Based Testing of Data Schemas.* In Proc. of the 19th Intl. Conference on Software Engineering and Knowledge Engineering, July 2007.

[10] FREEMAN, H.. *Software Testing*. IEEE Instrumentation & Measurement Magazine. Volume 5 Issue 3, pp. 48 – 50. September 2002.

[11] KAPFHAMMER, Gregory M.; SOFFA, Mary Lou. *A Family of Test Adequacy Criteria for Database-driven Applications*. In Proc. of the 9th European Software Engineering Conference, held jointly with 11th ACM SIGSOFT Intl. Symp. on Foundations of Software Engineering, Vol. 28 Issue 5, September 2003.

[12] NAZAR, I. F. *A Tool for Data Schemas Testing*. Master thesis, Computer Science Department, Federal University of Paraná. March 2007 (In Portuguese).

[13] OMG. *Meta-Object Facility Core Specification Version 2.0*. http://www.omg.org/ cgi-bin/doc?formal/ 2006- 01-01, January 2006. (accessed in September 2006).

[14] ROBBERT, M. A.; MARYANSKI, F. J.. *Automated Test Plan Generator for Database Application Systems*. In Proc. of the ACM SIGSAMLL/PC Symp. on Small Systems, pp 100-106, 1991.

[15] SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Database System Concepts*. 3rd ed., McGraw-Hill, 1998.

[16] SUÁREZ-CABAL, M. J.; TUYA, J.. *Using an SQL Coverage Measurement for Testing Database Applications*. In Proc. of the 12th Intl. Symp. on the Foundations of Engineering, November 2004.

[17] ZHANG, Jian; XU, Chen; CHEUNG, S.-C.. *Automatic Generation of Database Instances for White-box Testing*. In Proc. of the 25th Annual Intl. Computer Software and Applications Conference, pp 161 – 165, October 2001.

Analyzing Termination and Confluence in Active Rule Base via a Petri Net Approach

Lorena Chavarría-Báez, Xiaoou Li
Department of Computer Science
The Research and Advanced Studies Centre of the National Polytechnic Institute (CINVESTAV-IPN)
Av. Instituto Politécnico Nacional 2508, Col. San Pedro Zacatenco, Mexico City, C.P. 07360, Mexico
email: lchavarria@computacion.cs.cinvestav.mx, lixo@cs.cinvestav.mx

Abstract

Active rules allow software systems to execute actions automatically in response to events. Two desirable properties of active rule behavior are termination and confluence. In this paper we present a Petri net based approach to analyze termination and confluence in a rule base. Our approach is performed in two steps: first, we identify those rules which may not terminate and may not be confluent using their Conditional Colored Petri net representation. Second, we use semantic analysis to decide about termination and confluence of the detected rules. The great advantage of our approach is that it allows us to derive results about termination and confluence without analyzing all the rules.

1 Introduction

Active rules allow software systems behave automatically in response to inner or outer events. Using active rules, modern applications can be effectively developed. However, due to unstructured nature of rule processing it is quite difficult to predict the rule set behavior. Two desirable properties of active rule behavior are *termination* and *confluence*. A rule set is guaranteed to terminate if rule execution processing cannot continue indefinitely. A rule set is confluent if the final state is independent of the rule execution order.

Several works have been proposed in the literature to analyze termination and confluence. Some approaches tackled the problem based on triggering and execution graphs analysis [1], [2]. The triggering graph is directed graph whose nodes represent the rules and whose edges indicate that a rule produces an event that may trigger another rule. If the graph is acyclic, rule execution terminates. In execution graphs, nodes represent the state and directed edges are labeled with the name of the rule whose execution makes the system switch from one state to another. If the graph is acyclic and every pair of rules commute, the rule execution process is confluent. In [3] authors propagate the effect of the action part of a condition - action rule to the condition part of another rule to determine if the arcs of the triggering graph have to be included in the graph and to verify if two rules commute. In this work, all the rules have to be compared in pairs to accurately conclude about termination and confluence. In [4] authors translate a set of active rules into logical clauses and then apply results about termination and confluence available in the literature for deductive rules. In [5], confluence is investigated by using a rewriting technique. A user-defined transaction is translated by means of active rules into an induced one(s) and then they check their equivalence. This approach doesn't provide a general conclusion about confluence since its analysis is done on each initial user-defined transaction.

In this paper we present a Petri net based analysis approach for analyzing termination and confluence in an active rule base. One important aspect of our approach is that it avoids unnecessary rule analysis by identifying the Petri net structures of non-termination and non-confluence.

2 Active Rules

Generally, an active rule consists of three parts: an *Event*, a *Condition*, and an *Action*. So, they are also called ECA rules. The most common form of active rules is the following: **ON** *event* **IF** *condition* **THEN** *action.* An event is something that occurs at a point in time. The condition examines the context in which the event has taken place. The action describes the task to be carried out by the rule if the condition is fulfilled

once an event has taken place. From now onwards, we refer to an active rule as $R_i(E_i, C_i, Ai)$ where E_i, C_i and A_i are the event, condition and action of R_i, respectively.

Example 1. *Bank's policies for managing customers' accounts.*

An active database system (ADBS), which integrates active rule processing with traditional database functionality, is able to react automatically to meaningful events that are taking place inside or outside de database system. The following ADBS example is about a bank's policies for managing customer's accounts which is taken from [3]. It is based on the relation schemes `account(`*num, name, balance, rate*) and `low-acc(`*num, start, end*`)`, which contain information about bank's accounts, and a history of all time periods in which an account had a low balance, respectively. Policies are described below.

Policy 1: When an account's interest rate is modified, if that account has a balance less than 500 and an interest rate greater than 0%, then that account's interest rate is lowered to 0%.

Policy 2: When a new account is registered, if the account has an interest rate greater than 1% but less than 2%, then that account's interest rate is raised to 2%.

Policy 3: When a new account is registered, if that account has a balance less than 500 and is not yet recorded with a null end date in the `low-acc` relation, then the account is inserted into the `low-acc` relation with the current date as start and a null end date.

Policy 4: When a new account is registered, if the total number of low days for an account (as recorded in the `low-acc` relation) is greater than 50 and its current balance is between 500 and 1000, then its interest rate is set to 1% in the `account` relation.

Above policies are depicted as active rules as follows:

R1
`ON` update `account`.*rate*
`IF` update.*balance* $<$ 500 and update.*rate* $>$0
`THEN` update `account` set *rate* $= 0$
where *balance* $<$ 500 and *rate* > 0

R2
`ON` insert `account`
`IF` update.*rate* $>$1 and update.*rate* < 2
`THEN` update `account` set *rate* $= 2$
where *rate* > 1 and *rate* < 2

R3
`ON` insert `account`
`IF` insert.*balance* $<$ 500 and (not exists (select * from `low-acc` where `low-acc`.*num* $=$ insert.*num* and *end* is null))
`THEN` insert into `low-acc`(*num*, *start*, *end*) (select *num*, today( ), null from `account` where *balance* $<$ 500 and not exists (select * from `low-acc` where `low-acc`.*num* $=$ `account`.*num* and *end* is null))

R4
`ON` insert `account`
`IF` exists(select * from `account` where *rate* > 1 and *balance* $>$ 500 and *balance* $<$ 1000 and *num* in (select *num* from `low-acc` group by *num* having sum(*end-start*)$>$50))
`THEN` update `account` set *rate* $= 1$ where *rate* > 1 and *balance* $>$ 500 and *balance* $<$ 1000 and *num* in (select *num* from `low-acc` group by *num* having sum(*end-start*)$>$50)

Suppose the event "insert `account`" has been detected, then **R2**, **R3** and **R4** are triggered and their conditions must be evaluated. Let's suppose all conditions true, then, for simplicity, rule execution will be done by following the order of rules in the list. Therefore, **R2**'s action will be executed first, then **R3**'s action and **R4**'s action is executed finally. After **R2**'s action execution, the event "update `account`.*rate*" is signaled, so **R1** is triggered and execution process continues until there is no rule eligible to trigger. On the other hand, when **R3**'s action is executed, rule processing finishes since there is no rule triggered by the event "insert `low-acc`". Finally, when **R4**'s action is executed, the event "update `account`.*rate*" is raised and rule processing continues.

3 Active Rule Base Modeling

An active rule base as well as its interaction can be represented by the Conditional Colored Petri Net (CCPN) [6]. Unlike other graphical models, CCPN depicts each element of an active rule including composite events and condition evaluation.

As Figure 1(a) shows, an active rule is mapped to a CCPN structure as follows: a rule is mapped to a transition where its condition is attached, event and action parts are mapped to input and output places of the transition, respectively. Matching between events and input places has the following characteristics:

(1) *Primitive* places, represent primitive events;

(2) *Composite* places, represent composite events;

(3) *Copy* places, are used when one event triggers two or more rules. An event can be shared by two or more rules, but in Petri net theory, one token needs to be duplicated for sharing. A copy place takes the same information as its original one;

(4) *Virtual* places are used for accumulating different events that trigger the same rule. For example, to represent the composite event OR.

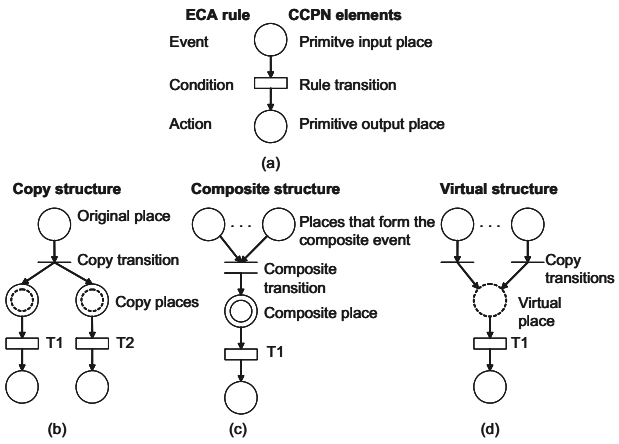
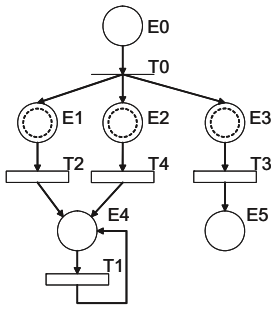


Figure 1. Basic CCPN structures of an active rule



Place [Type]	Event/Action
E0 [primitive]	insert account
E4 [primitive]	update account.rate
E5 [primitive]	insert low-acc
E1, E2, E3 [copy]	Insert account

Transition [Type]	Rule
T1[rule]	**R1**
T2 [rule]	**R2**
T3 [rule]	**R3**
T4 [rule]	**R4**
T0 [copy]	-

Figure 2. CCPN of the rule base of Example 1

Rules and transitions are related in the following form:

(1) *Rule* transitions, represent rules;

(2) *Composite* transitions, represent composite event generation;

(3) *Copy* transitions, duplicate one event for each triggered rule.

Whenever an event triggers two or more rules it has to be duplicated by means the copy structure depicted in Figure 1(b). Composite events formation is considered in CCPN using the composite structure drawn in Figure 1(c). Composite transition's input places represent all the events needed to form a composite event while its output place correspond to the whole composite event. Finally, we use the virtual structure to model the composite event `OR` as standing for in Figure 1(d). The CCPN model of a set of ECA rules is formed by connecting those places that represent both the action of one rule and the event of another rule. Figure 2 shows the CCPN model of the rules of Example 1. Correspondence between events/actions and places as well as matching between rules and transitions, are described in the figure too.

4 Termination and Confluence Analysis

Based on the CCPN model of a given active rule base we can analyze its termination and confluence properties. First, we represent the active rule base as a CCPN model and we identify the structures that may have non-termination and non-confluence problems. Since transitions in CCPN stand for rules, we actually obtain the rules that may exhibit abnormal behavior. Second, termination and confluence are investigated by analyzing rule semantic and rule interaction of rules computed during the first step.

Non-termination and non-confluence CCPN structures. In CCPN termination is depicted by cycles. If in the obtained CCPN model there is no cycles then rule processing is guaranteed to terminate. Otherwise, we will analyze the rules involved in the loop to ascertain if their processing finishes. Through CCPN model of Figure 2 we found the cycle formed by the sequence of places/transitions E4, T1, E4. So, **R1** in Example 1 triggers itself.

On the other hand, confluence problems appear when several rules are triggered at the same time since there are different ways to perform rule execution. In CCPN confluence is represented by the following CCPN structures: (1) copy typed transitions. In the CCPN of Figure 2 transition T0 is a copy typed transition, so, we need to analyze the execution order of transitions (rules) T2 (**R2**), T3 (**R3**), and T4 (**R4**). (2) rule typed transitions which evaluate the same condition. This kind of rules are identified in the CCPN by labeling them with the same index. Each rule in Example 1 evaluates different conditions, so, we don't need to analyze any transition of this type. (3) primitive places which have more than one input arc. The place E4 in the CCPN of Figure 2 has more than one input arc, so, we will examine the rule execution order of transitions (rules): T1 (**R1**), T2 (**R2**), and T4 (**R4**).

Rule semantic and rule interaction analysis. Once we have computed rules that may not terminate and may not be confluent, we analyze them taking into account their semantics and interaction. To analyze rule semantics, first we divide each rule action as following: (1) a modification operation, (2) a relation schema name, (3) attribute names of the relation schema, (4) new values for each attribute, and (5) a condition over attributes. We identify as Comm(A),

Table(A), Att(A), Val(A) and Cond(A) each one of the above elements, respectively. Let's see action A_1 = "update `account` set *rate* = 0 where *balance* < 500 and *rate* > 0". Its elements are: Comm(A) = update, Table(A) = `account`, Att(A) = rate, Val(A) = rate = 0, and Cond(A) = *balance* < 500 and *rate* > 0. Second, we compute the *target of a condition C*, denoted by Targ(C), which is the set of tuples which satisfies a condition C. In example 1, Tar(Cond(A_1)) ={t|t is a tuple and t.*balance* < 500 and t.*rate* > 0}.

Rules can interact in the following ways: activation, deactivation and commutativity. R_i can activate (deactivate) R_j if the execution of A_i makes condition C_j true (false) when it is evaluated. In our case R_i activates R_j if the following conditions are met:

(1) Comm(A_i) ="update" or "insert"
(2) Table(A_i)∩Table(A_j)≠ ∅
(3) $I \neq \emptyset$
(4) Val(I) ∈ Targ(Cond$_I$(C_j))
(5)Targ(Cond(A_i))∩Targ(C_j)≠ ∅
where I = Att(A_i)∩Att(C_j).

Let's consider rule **R1** of Example 1. Since A_1 and C_1 don't satisfy above condition (4), **R1** cannot activate itself.

Actions A_i and A_j commute, if for all database states, the execution of A_i followed by A_j (and vice versa) produce the same final database state. In our approach A_i and A_j commute if *Targ*(Cond(A_i))∩*Targ*(Cond(A_j))=∅. In Example 1 A_2 and A_4 don't commute since Targ(Cond(A_2))(={t|1<t.*rate* <2}) ⊂Targ(Cond(A_4))(={t|t.*rate*>0}).

Termination analysis is performed on the rule activation analysis. If in a cycle formed by rules R_1, R_2, ..., R_n ($A_n = E_1$) the rule R_k doesn't activate (or deactivate) the rule R_l, then the cycle finishes. Before, we found that **R1** in Example 1 triggers itself. However, it cannot activate itself. Then, its execution process finishes and rule base processing of Example 1 is guaranteed to terminate.

Conclusion about confluence is achieved by analyzing rule (de)activation and action commutativity. If simultaneously triggered rules don't (de)activate each other, and their actions commute, then the result of their processing is confluent and the rule base is also confluent. Through CCPN analysis we identify the rule sets which may trigger at the same time, namely {**R2**, **R3**, **R4**} and {**R1**, **R2**, **R4**}. Therefore, we need to analyze the following rule pairs: (**R2**, **R3**), (**R2**, **R4**), (**R3**, **R4**), (**R1**, **R2**), (**R1**, **R4**). If each one of those rule pairs is confluent, the rule base in Example 1 is confluent. Previously we have demonstrated that **R2** and **R4** actions don't commute, so, **R2** and **R4** are not confluent. In consequence, the rule base of Example 1 is not confluent.

5 Conclusion and future work

A Petri net based approach is proposed to analyze termination and confluence properties of an active rule base. Our approach has the great advantage of eliminating unnecessary rule analysis, since we don't need to analyze all the pairs of rules to draw conclusion about termination and confluence. In the future we will implement our approach so that the analysis could be done automatically.

References

[1] A. Aiken, J. Widom, and J-M. Hellerstein, Behavior of database production rules: termination, confluence, and observable determinism, *Proceedings of International Conference of ACM-SIGMOD,* pp. 59-68, 1992.

[2] E. Baralis, S. Ceri, and S. Paraboschi, Improved rule analysis by means of triggering and activation graphs, *Proceedings of the First International. Workshop on Rules in Database Systems,* Aug. 1993.

[3] E.Baralis, J. Widom, An algebraic approach to static analysis of active database rules, *ACM Transactions on Database Systems*, Vol. 25 , Issue 3, pp. 269-332, 2000

[4] S. Comani, L. Tanca. Termination and confluence by rule prioritization, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, No. 2, pp. 257-270, 2003

[5] D. Montesi, and R. Torlone, Analysis and optimization of active databases, *Data and Knowledge Enginnering*, Vol. 40, pp. 241-271, 2002.

[6] X. Li, J. Medina-Marín, S. Chapa, Applying Petri nets on active database systems, *IEEE Transactions on System, Man, and Cybernetics, Part C: Applications and Reviews,* Vol. 37, No. 4, pp. 482-493, 2007.

[7] N. Paton, O. Díaz, Active database systems, *ACM Computing Surveys*, Vol. 31, No.1, pp.62-103, 1999.

A Fuzzy Trigger Language for Relational Database Systems

Ying Jin Tejaswitha Bhavsar
Computer Science Department
California State University, Sacramento, CA - 95819, USA
jiny@ecs.csus.edu

Abstract - With the ever-increasing demands for data management, crisp data presentation, storage, and retrieval are not enough for complicated situations in real-life. Some kind of non-crisp or fuzzy semantics are desirable to better handle the situations where human judgment, evaluation and decisions are important. This project proposes a fuzzy language, named FZ-Trigger, for relational database systems. A trigger specifies the event raised by activities such as inserting a row, the condition to check upon event occurrence, and the corresponding action to perform when the condition is true. Incorporating fuzzy expression into the condition part of a trigger allows database users to specify the reaction to database events in a flexible manner. Uncertainty and imprecision factors are handled by linguistic variables. This paper describes the language specification of FZ-Trigger, the system architecture, and the implementation. A motivating example is also provided to illustrate the syntax and the use of FZ-Trigger.

1. INTRODUCTION

A relational database management system (DBMS) is based on the relational model that represents relations in a tabular form. Active database systems [1] are built on top of traditional passive databases by providing reactive services that automatically support monitoring and reacting to events. Active rules are the languages for active databases. The concept of active rules has been incorporated by modern commercial database systems, such as Oracle [2], in a simplified format of triggers. Triggers are database facilities that allow users to define the semantics of reactions with respect to different types of events. Triggers have a significant role in database systems for consistency control and business logic specification. A trigger consists of three parts, an event, a condition, and an action. The event is used to specify the origination of what happened, such as insert a row. The condition is evaluated upon the event occurrence to query over data sources. If the condition evaluation returns true, the action is performed to update the database or execute application procedures.

Triggers in traditional database systems are crisp, meaning that there is no vagueness and uncertainty. However, it is not sufficient to ever-increasing needs of data management. When users specify integrity constraints or business logics, they are limited to use precise expressions. A new language to use fuzzy semantics, named FZ-Trigger, is proposed in this paper to incorporate fuzzy concepts into database triggers. FZ-Trigger can be used over traditional crisp databases. FZ-trigger allows relational database users to define triggers using fuzzy conditions over crisp data.

For example, in traditional non-fuzzy trigger, we can specify "*When an item is sold, check the quantity of item in stock. If the quantity is less than 20 and the popularity rating of the item is 90, place a purchase order*". In the FZ-Trigger system, we can define "*if the item quantity is **less** and popularity is **high**, place a purchase order*". People are more familiar with this type of expression in natural language. Moreover, it is reasonable that cases such as "item quantity is 19 and popularity is 88" (in addition to "item quantity is 20 and popularity is 90") are also considered to cause the action to be performed. Our trigger specification is consistent with SQL3 [2]. It is well-known that different relational database vendors provide different relational database implementations with small variation from SQL standard. This paper uses the popularly used DBMS Oracle as the implementation system to illustrate the use of FZ-triggers.

The paper is further organized as follows. Section 2 covers the related work in the areas of fuzzy triggers and fuzzy relational database systems. Section 3 illustrates the motivating example used in this work and describes fuzzy concepts. Section 4 describes the FZ-trigger language. Section 5 presents the system architecture and implementation. We conclude the paper with summary and future directions in Section 6. The non-fuzzy trigger is referred to as crisp trigger in this paper.

2. RELATED WORK

Applying fuzzy concepts to database systems has been a research topic over years, such as in [4] [5],

including how to add fuzziness in the stored data as well as how to process fuzzy queries. Galindo [6] proposed a fuzzy query language called FSQL for fuzzy relational databases. FSQL is an extension to the SQL query language standard to allow flexible queries. Limited research has been done on triggers so far. C-fuzzy and CA-fuzzy triggers are proposed by [7] and [8] to incorporate fuzziness into active database system named as TEMPO. C-fuzzy triggers are limited to its active database system for a specific control system. Neither the language nor the architecture design can be applied to other systems easily without major modifications. In this proposed project, we will specify fuzzy triggers based on the trigger specification of SQL3 standard. We also incorporate the concepts of fulfillment of thresholds to the conditions using linguistic variables. The existing relational database retains the crisp values, which allows all existing database applications unaltered.

3. MOTIVATING EXAMPLES AND FUZZY CONCEPTS

To illustrate our approach of incorporating fuzziness into triggers, we use a motivating example of Fashion Store Inventory, in which the store maintains information such as product, price, item-code, quantity available, and category. It keeps track of the transaction history after each item is sold. Discount offer information, yearly/monthly sales of each product and order placement priority is also maintained. Purchase orders are placed for the items which are popular and less in quantity. Items which have high quantity left in the inventory can be declared with a clearance discount or seasonal discount. This is a typical database application that stores crisp information in all the tables. In this paper, we present our approach of specifying fuzzy triggers over crisp underlying data.

An important concept in fuzzy theory is linguistic variables, whose values are words rather than numbers. The linguistic variable is represented by a quintuple that characterizes the fuzzy number along with the linguistic concepts interpreted in a particular context.

The quintuple is **<v, T(v), X, g, m>**, where *v* is name of the linguistic variable, *T(v)* is a set of linguistic terms applicable to variable *v*, *X* is a universal set of values, *g* is grammar for generating the linguistic term, and *m* is the semantic rule that assigns to each term. For example, for a linguistic variable *Quantity,* the set of linguistic terms could be T (*Quantity*) = {**Very_Low, Low, Sufficient, High, Very_High**}. Among many other presentations, trapezoidal distribution is chosen to use in this research. For example, as shown in Figure 1, the linguistic term, **Low,** for linguistic variable *Quantity,* is represented using trapezoidal function as **Low** (8,10,16,18), where $\alpha = 8$, $\beta = 10$, $\gamma = 16$, and $\delta = 18$.

To specify fuzzy triggers over crisp data in our implementation, we create an additional table to store information related to linguistic variables, without altering any existing tables to avoid the effect on existing applications. We also store α, β, γ, and δ values of each trapezoidal distribution in this table. The table contains the information about linguistic variable (e.g. **Low**), the name of table (e.g. *Item* table) related to this linguistic variable, the attribute name (e.g. *Quantity*) related to this linguistic variable, as well as the values for α, β, γ, and δ. The table serves as the metadata for fuzzy knowledge to allow other architectural components to retrieve fuzzy knowledge at run time.

Fulfillment threshold specifies the degree $d \in [0, 1]$ for the specification of a condition. For example, we can specify a grade of 0.75 (the variable named as THOLD) when we state a condition like "*Quantity* is **Low**", as shown in Figure 1, which eventually forms a range of (9.5, 16.5). Using **Low** instead of the range of (9.5, 16.5) to describe the *Quantity* is more near to nature languages used by human beings.

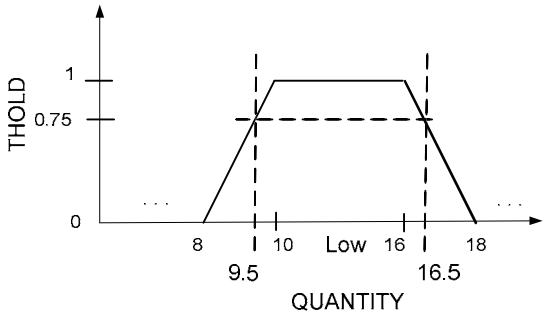


Figure 1: Example to show trapezoidal function

4. LANGUAGE

This section describes the language and example of FZ-Trigger. FZ-Triggers are defined as:

$$\textbf{\textit{when E, if }}\{C \times S_i \times S_j\}\textbf{\textit{, then A}}$$

When event occurs, if the condition is true, then the action is performed. In this definition, ***C*** is a set of fuzzy conditions connected by *AND* and *OR*. Each condition in ***C*** consists of two elements (v Θ g, d), where ***v*** is a linguistic variable defined in the quintuple in Section 3. ***Θ*** is a comparison operator which includes =, <, <=, >, >=, !=. ***g*** is linguistic term where ***g ∈ T(v)***, ***T(v)*** is defined in the quintuple in Section 3. ***d*** is fulfillment threshold which applies to "v Θ g", ***d*** ∈ [0, 1]. S_i and S_j are the current and previous database states, which are referred as *new* and *old* in triggers.

The syntax of the proposed FZ-Trigger language is shown in Figure 2. A trigger consists of three parts: event, condition, and action. The event is the data manipulation command, such as Insert. The condition part of the trigger can be specified in two places (bold in Figure 2): 1) In the "WHEN <condition>", 2) In the "IF <condition>" within a PL/SQL statement. Both crisp and fuzzy expressions can be specified in these two places (Crisp expressions are also allowed in FZ-trigger). The action part of the trigger is specified in the PL/SQL statement. In traditional crisp triggers, users can specify an expression such as "*new*.*Quantity* = 8". In FZ-Trigger, users can alternatively specify a fuzzy expression such as "*new*.*Quantity* = **$Low**" along with the specification of fulfillment threshold such as "WITH THOLD = ***0.8***". As a result, users specify the semantic of "Quantity is high with the degree of 0.8" using the fuzzy expression. We use '**$**' to identify fuzzy values, and 'THOLD' to indicate a fulfillment threshold. 'THOLD' is optional and the default value taken is ***1*** if nothing is specified. Multiple fuzzy expressions can be connected by *AND* or *OR*.

An example of FZ-trigger is illustrated in Figure 3, which is based on the motivating example. In this example, we specify the fuzzy condition in the "IF" statement within a PL/SQL block (in bold). The trigger specifies the constraints that when the *Quantity* of an item is **Low** with the fulfillment threshold of ***0.8***, and when the *Popularity* is **High** with the degree of ***0.7***, we should create a pending order if we have not already placed a purchase order for this item. The trigger also updates the *Item* table to maintain the consistency between *Item* table and *Transaction_History* table.

```
CREATE TRIGGER trigger name
(AFTER | BEFORE) triggering events ON table name
[FOR EACH ROW]
[WHEN [crispExpression] [[and|or] [fuzzyExpression [with THOLD =
TholdValue]]]*]
BEGIN
  PL/SQL Block
  [IF [crispExpression] [[and|or] [fuzzyExpression [with THOLD =
TholdValue]]]*]
  PL/SQL Block
END;
/
```

Figure 2: Syntax of FZ-Trigger

```
CREATE OR REPLACE TRIGGER ON_SOLD_PENDING_ORDER
AFTER INSERT ON TRANSACTION_HISTORY
REFERENCING NEW AS NEW
FOR EACH ROW
DECLARE
   ItemObj   Item%rowtype;
   Count1    Number;
BEGIN
   select * into ItemObj from Item I where I.Item_ID = :new.Item_ID;
   select count(*) into Count1 from Pending_Orders where Item_ID =
:new.Item_ID;
   IF(ItemObj.Quantity = $Low WITH THOLD = 0.8) AND
     (ItemObj.Popularity = $High WITH THOLD = 0.7) THEN
   if Count1 = 0 then
      insert into Pending_Orders values(:new.Item_ID, 21, 'High');
   end if;
   end if;
   update Item set Quantity = Quantity - :new.Quantity,
                   Popularity = Popularity + :new.Quantity
                   where Item_ID = :new.Item_ID;
END;
/
```

Figure 3: Example of FZ-Trigger

5. ARCHITECTURE AND SYSTEM IMPLEMENTATION

The system architecture is shown in Figure 4. A FZ-Trigger system has been implemented to allow users to specify fuzzy triggers. The *User Interface* provides an interface for users to enter a fuzzy trigger and view the output. This interface passes the fuzzy trigger creation request to the *Coordinator*. Then the *Parser* parses the given fuzzy trigger, consults the metadata for fuzzy knowledge, and then translates the fuzzy trigger into a crisp trigger. The parser is implemented using JavaCC [9]. Using *JDBC* (Java Database Connectivity), the newly generated trigger is passed to the underlying database. The result (e.g. the fuzzy trigger is successfully generated) is passed to the *Coordinator* and then to the users through the *User Interface*. For an existing DBMS, the execution model of crisp triggers has already been established. We utilize the existing DBMS environment for the execution of the newly generated trigger at run time.

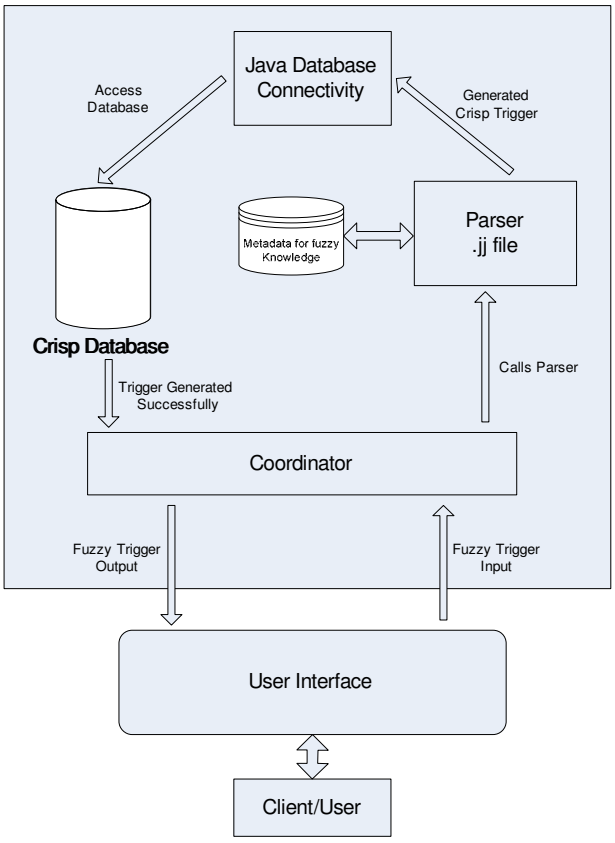


Figure 4: Architecture of FZ-Trigger System

6. SUMMARY AND FUTURE DIRECTIONS

To support vague expressions over crisp data in databases, we developed the FZ-Trigger language that supports fuzziness in relational database triggers. Fuzzy triggers can be applied in various real-world applications to allow flexible expressions of business logic. This paper describes the syntax and use of the language, the system architecture, and the implementation details. One of our future directions is to provide better interface to allow users to express FZ-Trigger easily with minimum requirements of syntax. Another direction is towards the evaluation of the performance of the FZ-Trigger system.

REFERENCES

[1] N. W. Paton and O. Diaz, "Active database Systems," *ACM Computing Surveys,* 31,1, March, 1999, pp. 3-27.

[2] Oracle Database, http://www.oracle.com/database/index.html.

[3] P. Gulutzan and T. Pelzer, "SQL-99 Complete Really", Miller Freeman Publishing, 1999.

[4] K. L. Joy, S. Dattatri, "Implementing a fuzzy relational database using community defined membership values," *Proceedings of 43rd ACM Southeast Conference*, Kennesaw, GA, March18-20, 2005.

[5] D. Li and D. Liu, "A Fuzzy Prolog Database System", Taunton, England: Research Studies Press, 1990.

[6] J. Galindo, J. Medina, O. Pons and J. Cubero, "A Server to Fuzzy SQL Queries", In "Flexible Query Answering Systems", T. Andreasen, H. Christiansen and H.L. Larsen (Eds.). Lecture Notes in Artificial Intelligence (LNAI) 1495, Ed. Springer, 1998, pp. 164-174.

[7] T. Bouaziz, J. Karvonen, A. Pesonen, and A. Wolski, "Design and Implementation of TEMPO Fuzzy Triggers," *Proceedings of Eighth International Conference on Database and Expert Systems Applications*, Toulouse, France, September, 1997, pp.91-100.

[8] A. Wolski, T. Bouaziz, "Fuzzy Triggers: Incorporating Imprecise Reasoning into Active Databases," *ICDE*, *14th International Conference on Data Engineering*, 1998, p. 108.

[9] Introduction to JavaCC, www.engr.mun.ca/~theo/JavaCC-Tutorial/javacc-tutorial.pdf.

A Comparative Study on Data Representation to Categorize Text Documents

D.A. MEEDENIYA [1], A.S.PERERA[2]
[1]Department of Computer Science and Engineering, Faculty of Engineering, University of Moratuwa, Sri Lanka,
[1]dulani1@yahoo.com
[2]Department of Computer Science and Engineering, Faculty of Engineering, University of Moratuwa, Sri Lanka,
[2]shehan@cse.mrt.ac.lk

Abstract - **In the modern world text documents play an important role in most of the organizations. Their constant growth widens the scope of document storage. As a result, there is a potential need for effective text retrieval and search capabilities. This paper suggests two document preprocessing methods. The objective of this study is to find an appropriate data representation for text categorization by comparing two data representation approaches. The first approach groups the documents based on their title and the second approach considers the document body to group documents. Both methods apply the same clustering and classification techniques on the test data sets. It applies clustering to divide the documents into categories and uses classification techniques to validate the clustering results. This study shows that the text documents grouping based on document titles has high performances than the other approach.**

Keywords — Title data representation, Document body data representation, K-mean clustering, Naïve Bayes classification, WEKA data mining tool.

I. INTRODUCTION

Information has a strategic importance for most of the organizations as well as for every citizen. The constant growth of text documents widen the document storage scope and make it difficult to retrieve information. The growing importance of electronic media for storing and exchanging text documents has led to an increasing interest in tools and approaches for dealing with information included in the text documents. Therefore there is a requirement for advance analysis of those documents to group them according to the content as well as semantic search and document comparisons.

Text categorization is the process of grouping documents into classes based on their content. It assigns unseen documents into categories and handles the exponential growth in available text documents [1]. Text mining extracts high quality information from text documents by considering the patterns in the data set. Patterns are found when there are relationships among data sets. A careful study of the business case and the correlation of the data sets have to be considered to discover a pattern.

When text clustering is carried out manually, the analysts need a special knowledge in vocabulary and knowledge processing. In the categorizing process the analysts have to read the full text document, memorize all class definitions and also it consumes more time. Today most of the applications such as news dispatching and e-mail filtering deal with a large collection of documents and it is difficult to cluster them manually based on their content. By using an automatic text categorizing system it is possible to cluster documents more accurately within a short period of time.

Data sets are rich with hidden information that can be used for making intelligent decisions. Most of the text mining and information retrieval techniques rely on word matching. Clustering can be used as an alternative technique for information retrieval. Clustering is the process of partitioning of the data sets into subsets, so that the data in each cluster share common characteristics. Clustering gives an overview of a document collection, manages a large number of text documents and provides efficient information retrieval [1]. Classification and prediction can be used to extract models describing important data classes or predict future data trends, where as visualization has the ability to describe the structure of a classifier in an understandable way by converting the data into usable knowledge [2].

Most of the full text documents are rather long and potentially with a loose structure. Previous studies show that word clusters can reduce the feature space dimensionality, with only a slight change in classification precision [3]. Most common approaches start by evaluating the co-occurrence of words versus documents [1]. However the count matrices tend to be sparse and noisy when the data set is relatively small. Although the documents are represented in a high dimensional sparse feature space, it is not optimal for classification algorithms [4]. Considering all practical settings it is difficult to apply clustering in a high dimensional space. Furthermore it is hard to explain well, why the text clusters have been constructed the way they are [5]. It is difficult to obtain a categorization that is both meaningful and complete [5].

This paper considers the problem of categorizing abstracts, which describes the NSF (National Science Foundation) awards for basic research. This study is based on the comparison of two document categorization approaches. One

approach categorizes documents based on the title of the article, where as the other is based on the article body. These approaches use both clustering and classification techniques to extract knowledge, based on the content of the documents. The objective is to show that the approach based on article titles, gives better performance than the approach based on article body. The study finds an acceptable data preprocessing method which can be used to categorize similar documents into one category.

This paper, describes the data set, tools and techniques used in this study. Next it explains the methodology and finally it includes the results obtained and concludes the results.

II. DATA SETS

Data sets are rich with hidden information that can be used for making intelligent decision. The selected data set for this study is publicly available at UCI Knowledge Discovery in Databases archive. The 'nsfabs', (NSF abstracts), data set consists of 129,000 abstracts, one per file. It describes the NSF awards for basic research during the period 1990-2003 [6]. For simplicity we used only 500 abstracts for testing.

III. RESEARCH METHODOLOGY

A. Outline

This approach used word frequency counts to determine the significant words in each document. Every word is represented only once in the text representation.

Data consists of textual data with noise; hence some attributes may be irrelevant to the clustering and classification tasks. Since the data is extracted from raw data files, relevance analysis on data was performed to remove redundant attributes from learning process. A data cleaning and preprocessing algorithm was used to avoid the interference of irrelevant words during the clustering by extracting, removing noise and cleaning the textual data. After applying cleaning on the original data set, it is easy to extract words with a high occurrence, which leads to the main idea represented in the document. We have recognized and classified significant vocabulary items from the text.

B. K-Means clustering algorithm

The K-means algorithm is simple to use and can run on large data sets. It is a widely used central clustering technique that minimizes the average distance between an observation and its cluster center. The algorithm first selects the number of clusters and determines the cluster centers. It assigns each collection of words to the nearest cluster center by minimizing the average squared Euclidean distance between the words and its cluster center and re-computes the new cluster centers.

This procedure is repeated until some convergence criterion is met [2], [1].

C. Waikato Environment for Knowledge Analysis (WEKA) data mining tool

This is an easy to use, extensible package which contains a collection of machine learning algorithms for solving real world data mining problems. It is written in JAVA and runs on most platforms. Implemented schemes for classification, numeric prediction and meta-schemes are three main schemes in WEKA [9].

D. Classifiers

Naïve Bayes is a simple, popular text classification algorithm which classifies using estimator classes. This allows for a detailed analysis of the effects of using word clusters instead of words as features [4]. It uses joint probabilities of words and categories to estimate the probabilities of categories in a given document [10].

J48 classification algorithm generates a C4.5 decision tree. A decision tree builds classification models to predict classes for unseen entities. It is a simple structure where non-terminal nodes represent tests on one or more attributes and terminal nodes reflect decision outcomes [9].

E. Test Options

Following test modes have applied with classifiers.

i. Use training set: evaluates on how well it predicts the class of the instances it was trained on.

ii. Cross-validation. The classifier is evaluated by cross-validation, using the number of folds that are entered in the Folds text field.

iii. Percentage split: evaluates on how well it predicts a certain percentage of the data which is held out for testing [9].

F. Methodology:

i. Acquire text documents: Gathered text document in a common format and name the text files in a standard format in a suitable way which can be used for the data preprocessing task.

ii. Data cleaning: Applied a data cleaning and noise removal algorithm which extracts only the necessary words from each text file. The algorithm eliminates prepositions, symbols; numeric form of numbers, proper names, punctuation marks, commonly used words, abbreviations and other non-alphabet characters.

iii. Data pre-processing: This study experimented with two approaches. The first approach extracts the first five words from the title without considering any other content in the document. The extracted words are stored in one file, so that each row represents the keywords relevant to one document title. The second approach counts the number of occurrences of each word in the document body and extracts five words with the highest frequency, from each document. All extracted words were stored in a matrix, so that each row represents the words with the highest frequency for a given document.

iv. Document categorization: Applied K-means clustering algorithm on the pre-processed data and identified different clusters, which each document belongs to. The experiment

was done with both 35 and 50 clusters. The number of clusters is decided based on the number of available documents.

v. Document classification: Applied three classifiers Naïve Bayes and J48 on the categorized data to verify the precision of the categorization in order to compare the two data representations suggested in this work.

vi. Analyze results: Computed the mean, median, maximum, minimum, and standard deviation of the obtained results to analyze clustering process. Next we compared the cluster distribution and the accuracy of the two approaches and identified the better approach.

IV. EXPERIMENTAL RESULTS

This study considered several criteria to evaluate the quality of clustering. The results obtained using WEKA tool, after classification according to the number of clusters in both data representations are shown in the Table 1 and Table 2.

According to the results it is not necessary to increase the number of clusters to improve the accuracy, because in some cases the accuracy with 35 clusters is better than the accuracy with 50 clusters.

A. Test 1: Data representation using document titles

TABLE 1
Results obtained by applying classifiers on the data set which is clustered using K-means technique.

Technique	Option	With 35 clusters		With 50 clusters	
		Accuracy	Mean absolute error	Accuracy	Mean absolute error
Naïve Bayes	Using training set	92.4%	0.0109	91.2%	0.0086
	Cross validation (with 10 folds)	60%	0.0299	63.4%	0.0223
	Percentage Split 66%	62.3%	0.0319	58.2%	0.0237
J48	Using training set	87.6%	0.0105	85.2%	0.0087
	Cross validation (with 10 folds)	71.2%	0.0274	66.4%	0.0213
	Percentage Split 66%	67%	0.0285	59.4%	0.0224

B. Test 2 : Data representation using document body

TABLE 2
Results obtained by applying classifiers on the data set which is clustered using K-means technique

Technique	option	With 35 clusters		With 50 clusters	
		Accuracy	Mean absolute error	Accuracy	Mean absolute error
Naïve Bayes	Using training set	99.6%	0.0016	99.8%	0.001
	Cross validation (with 10 folds)	43.4%	0.0438	43.2%	0.032
	Percentage Split 66%	40.5%	0.0458	32.3%	0.0341
J48	Using training set	43.6%	0.0454	34.4%	0.0347
	Cross validation (with 10 folds)	43.6%	0.0454	35.2%	0.0342
	Percentage Split 66%	40%	0.0459	32.3%	0.0349

According to the results Naïve Bayes classifier gives high accuracy with the 'Training set data' option compared to other tests. Moreover, the 'Title' data representation gives better quality, more than 58%, in almost all the cases when compared to the data representation based on the 'Document body'.

C. *Analysis of cluster Performance*

Considering all the 12 accuracy results for each representation separately, Table 3 depicts the analysis results for the classification process.

TABLE 3
Analysis of Results

Statistics on Accuracy	Title Data Representation	Document body Representation
Max	92.40%	99.80%
Min	58.20%	32.30%
Mean	72.03%	48.99%
Median	66.70%	41.85%
Standard Deviation	13.214	24.081

As the results depict, 'Title' data representation has a higher value for the mean, and a lower value for the standard deviation than the 'Abstract' data representation. It can be seen that the 'Title' data representation leads to cluster the data better than the 'abstract content' representation.

V. DISCUSSION

We compared two data representation approaches that can be used to divide text documents into groups. By extracting words that maximize the information about the documents, we could obtain low dimensional representation of the documents. On the basis of the experimental results it can be seen that the K-Means clustering algorithm and Naïve Bayes classification algorithm can be used to group the text documents with significant accuracy. Moreover the data representation plays an important role in the clustering.

According to the results 'Title' data representation is important because it captures the main idea of the document. The document body may not include words with high frequency, which give the main idea of the text. We considered proper names, multiword terms, abbreviations, and other useful things such as numerical forms of numbers, percentages and money. Moreover the document body can contain synonyms, phonemes and words with different tenses which may not account for the word frequency in this study. This can be enhanced by defining some test cases manually and by training the data set based on the test cases.

VI. CONCLUSION

Large collection of documents may afford a lot of useful information to people. However, it is a challenge to find out the useful information from a large collection of documents. Successfully implemented text mining techniques help to identify the category of a text document, which it belongs to. The results of this study are generally applicable to any domain with textual data and helps when dealing with large amount of data. It verifies the clustering results by applying classification on clustered data. Moreover it is not necessary to increase the number of clusters to gain better results. The data representation in this study enables to improve the clustering quality. However the clustering approaches which are based on frequency of terms may not exhibit significant structural information as their data points are not similar to each other. The data preprocessing step guarantees completeness by solving this problem.

This approach can be practically applied to group news articles according to their subject and for other applications such as analyzing insurance claims, fax processing, e-mail filtering.

ACKNOWLEDGEMENTS

This research was carried out as a partial requirement for the M.Sc. in computer science, University of Moratuwa.

REFERENCES

[1] Sameh H. Ghwanmeh, "Applying Clustering of Hierarchical K-means-like Algorithm on Arabic Language", *International Journal Of Information Technology,* Volume 3 Number 1 2006 Isbn 1305-2403.

[2] Jiawei Han and Micheline Kamber, *Data Mining: Concepts and Techniques,* Second Edition, Morgan Kaufmann publications, United States of America, 2006.

[3] L. Douglas Baker, Andrew Kachites McCallum, "Distributional Clustering of Words for Text Classification", *ACM SIGIR 98*, 1998.

[4] Noam Slonim, Naftali Tishby, "The Power of Word Clusters for Text Classification", *23rd European Colloquium on Information Retrieval Research*, 2001.

[5] Andreas Hotho, Alexander Maedche, teffen Staab, "Ontology-based Text Document Clustering", Text Learning: Beyond Supervision Workshop, CAI 2001.

[6] The UCI KDD Archive, Available at http://kdd.ics.uci.edu/databases/nsfabs/nsfabs.html .

[7] Bei Yu, John Unsworth , "An Evaluation of Text Classification Methods for Literary Study".

[8] Hannes Wettig, Jussi Lahtinen, Tuomas Lepola, Petri Myllym¨aki and Henry Tirri, "Bayesian Analysis of Online Newspaper Log Data", *Proceedings of the 2003 Symposium on Applications and the Internet Workshops*, California, 2003, Pp. 282–287.

[9] Waikato Environment for Knowledge Analysis (WEKA), version 3.4.11, University of Waikato, New Zealand.

[10] Y.C. Fang, S. Parthasarathy, F. Schwartz, "Using Clustering to Boost Text Classification", Ohio State University.

[11] M. Bernotas, Kazys karklius, remigijus laurutis, asta slotkienė , "The Peculiarities of The Text Document Representation, Using Ontology And Tagging-Based Clustering Technique ", *Information Technology And Control*, 2007, Vol.36, No.2 .

[12] David Kirk Evans Judith L. Klavans, Kathleen R. McKeown, "Columbia Newsblaster: Multilingual News Summarization on the Web", Department of Computer Science, Columbia University, NY.

An Example on Economics-Driven Software Mining

Rami Bahsoon
School of Computer Science,
The University of Birmingham,
Edgbaston, Birmingham B15 2TT, UK
r.bahsoon@cs.bham.ac.uk

Wolfgang Emmerich
London Software Systems, Dept. of Computer Science,
University College London, Gower Street, WC1E 6BT, UK
w.emmerich @cs.ucl.ac.uk

Abstract

Economics-driven software mining (EDSM) sifts through the repository data to extract information that could be useful for reasoning about not only the technical aspects but also the economics properties related to the development and/or evolution of software systems, and in relation to the environments in which they are procured, developed, evolved and used. The objective is to provide the analyst with insights into investment decisions related to the development, maintenance, and evolution of software systems. We describe a scenario for realizing EDSM through an example. The example represents a small-size component-based distributed architecture, where we mined existing performance repositories to value the ranges in which a given software architecture can scale to support likely changes in load. The mining is based on a financial analogy. The mining step is then complemented with real options analysis to predict the values resulted from the ranges in which an architecture can scale under uncertainty, where uncertainty is attributed to the unpredicted change in load. The example shows the usefulness of EDSM in informing tradeoffs analysis in software design decision making.

1. Introduction

Effort on Mining Software Repositories (MSR) [MSR 1-4] has revolved around approaches which analyze the data stored in software repositories to assist in program understanding and visualization; predict and gauge the reliability and quality of software systems; study the evolution of software systems through discovering patterns of change and refactorings; modeling defects and their repair; and understand the origins of code cloning and design changes. Contributions have also included case studies showing how data can be extracted from software repositories to improve software design and reuse. The overall goal is utilize the mined data for predicting and planning various aspects of software projects. Meanwhile, software engineers are faced with general lack of adequate models and methods, which connect technical engineering concepts to economics and value creation under given circumstances [EDSR 1-8]. Reflecting on the Software Engineering discipline, [Sul99] note that the problem in the field is that "no serious attempt is made to characterize the link between structural decisions and value added". That is, the traditional focus of software engineering is more on structural and technical perfection than on value added [EDSR 1-8; Boe00; Erd00]. This argument is applicable to the emerging MSR discipline, where the current focus appears to be purely a technical endeavor with little attention paid to economics context. For example, software repositories are often mined and analyzed ignoring the link between technical properties, economics, and value creation under a given circumstances. Such a link may provide the software analyst with a powerful tool for predicting cost/value information for developing and evolving dependable software; understanding the economics of refactoring and reengineering; assisting in resource planning and utilization; and understanding the economics ramification of the change; defects and their repair; on the system and its design artifacts (e.g., architectures); and informing design trade-offs. The objective is to utilize data buried in software repositories to provide insights into investment decisions related to the development and evolution of software systems to assist in resource planning and utilization. Conversely, mining software repositories could be seen as an effort for empirically developing economics-driven software engineering models and methods, which could have the promise in addressing the need indicated by [Sul99; Boe00; EDSR 1-8].

In this paper, we describe a scenario for realizing EDSM. Drawing on a case study that adequately represents a medium-size component-based distributed architecture, we mined existing performance repositories to value the ranges in which a given software architecture can scale to support likely changes in load. The mining is based on a financial analogy, where we utilize the concept of twin asset in financial engineering to justify mining relevant repositories. The mining process in then complemented with real options analysis for predicting the values resulted from the ranges in which an architecture can scale under uncertainty, where uncertainty is attributed to the unpredicted change in load. As the exact method for analyzing scalability is subject to debate, we focus the analysis on *throughput* as a way for measuring scalability. The provided pointers describe how EDSM can inform tradeoffs in software design decision making.

The paper is further structured as follows. Section 2 defines EDSM. Section 3 presents an example on realizing EDSM. Section 4 briefly outlines related work. Section 5 concludes.

2. Economics-Driven Software Mining

Economics-driven software mining (EDSM) is based on the premise that non-trivial, unknown, and valuable information lies in an existing data repository, where the goal of the mining is to sift through the repository data to extract information that could be useful for reasoning about not only the technical aspects but also the economics properties of the development and/or evolution of software systems, with the environments in which they are procured, developed, evolved and used [Bah07].

According to [Min99], the process of mining software repositories encompasses: (i) *Data extraction* from repositories;

(ii) *preprocessing the data for analysis*, where the extracted data has to be formatted (e.g., treating noisy or missing data), sampled, and often need to be adapted to the mining algorithm(s). The data is then ready to be mined by a data mining algorithm(s); (iii) *data mining* which aims at extracting patterns of interesting and potentially useful, unknown, non-trivial information from the data; and (iv) *data interpretation* where the patterns identified are interpreted into knowledge, which can then be used to support decision-making. Different mining techniques may be used to achieve this step. EDSM poses several challenges. For example, how can we decide on which data to be extracted and be mined? Which of the mined data could be revealing to both the technical and economical properties of a software system and relative to the mining objectives? What are the mining tools that could be used for extracting meaningful inputs for the economics-driven software engineering analysis? How can we ensure that the mining objectives have been satisfied and the obtained knowledge is meaningful inputs to the economics-driven software engineering analysis? What are the appropriate analyses tools that could be used for supporting EDSM? The challenge, therefore, is to realize EDSM in light of these questions.

The process of mining software repositories includes (i) setting a goal for the analysis (i.e. the mining objective); (ii) selecting the economics models which can perform the analysis; (iii) developing the mining tools for extracting and mining information, which could serve as inputs for the models in (ii); (iv) capturing and interpreting the derived patterns; (v) modeling and computation; and (vi) result interpretations, analysis and reflection, where the mining step can be complemented by economics analysis to provide an answer for queries related to the economics of software artifacts, project utilization, and management. The queries could range from simple to compound ones. For example, let us assume that the query is to understand the evolution pattern of component X in a given architecture and the cost trends of evolving X over a time period. The change history of X could be mined using existing approaches and can be then complemented by cost estimation to cast effort of evolving X over a given period to cost (in £). Note, these models could be adapted from finance, economics, etc. on condition that the model assumptions are plausible or simplified to serve the software engineering mining objectives. The drawn analogy, the model inputs, and the made assumptions can then justify mining relevant repositories. In some cases, the analysis tool tends to shape the mining tool. An example is provided in Section 3, where adopting options analysis from financial engineering has constrained the way we extract and mined data.

3. An Example

We provide an example on realizing EDSM from our application of real options theory in software engineering [Bah08].

Setting. Let us consider a three-tier architecture of an online banking system application, referred to as Duke's. This architecture will be built on middleware, such as Java 2 Enterprise Edition (J2EE) and the Common Object Request Broker Architecture (CORBA). Depending on which middleware is chosen, different architectures may be induced [DiN99]. Given the choice of either CORBA or J2EE to induce an architecture, let us assume that the Duke's Bank system needs to scale to accommodate the growing number of clients in one-year time. An architecture which can scale to address such changes in load with limited resources and shorter time-to-market is a significant asset for surviving the business, cutting down maintenance costs, utilizing resources, and creating value. In particular, the cost and value derived from *the flexibility* in scaling up due to inducing the architecture with either CORBA or J2EE can inform the decision tradeoffs in considering either. Hence, the value added can inform the selection of application server products to induce Duke's. We show how existing *performance benchmark repositories* are utilized and mined to predict the values in which Duke's architecture, when induced by each middleware, can scale to support changes in load. We mined relevant performance benchmarks to understand how the architecture of the system may behave once induced with either and with respect to throughput, which is a scalability and load measure. The mining is based on a financial analogy, where we "mimic" the concept of *twin asse*t in financial engineering to justify minin*g* relevant repositories and for valuing throughput using historical data. The approach utilizes online data and benchmarks, submitted from different practitioners and vendors. The mining process is then complemented with real options analysis for predicting the values resulted from the ranges in which an architecture can scale under uncertainty, where uncertainty is attributed to the unpredicted change in load. The rationale is that the combination could provide the architect/analyst with a useful tool for understanding the extent to which the software system is can accommodate the change in load and starting from early stages of the software lifecycle, where the system need not be implemented.

Setting the mining objectives. Let us assume that we are given the choice of two middleware M_0 and M_1 to induce the architecture of a particular system as it is the case of Duke's. Let us assume that S_0, S_1 are the architectures obtained from inducing M_0 and M_1 respectively. Say, M_1 is an economical choice, if it adds value to S_1 relative to S_0. We attribute the added value to the enhanced flexibility of S_1 over S_0 in scaling up the architecture. But the added value is uncertain, as the demand and the nature of the future change and load are uncertain. We set some queries: (i) How valuable is the flexibility of either alternative, relative to likely change in scalability, will be in the long-run? (ii) Which solution is more valuable under uncertainty, where uncertainty is attributed to the unanticipated changes in load? (iii) What is the impact of volatility on value creation under given consideration? (iv) What is the impact of uncertainty on our choice? (v) Can high uncertainty, due to the likely future load, make the less favorable technology more appealing for the decision maker (and vice versa)? The challenge now is to select the economics model(s), which could be suited for addressing the said objectives. The value of flexibility under uncertainty is critical to choice of the economics models.

Selecting economics models, which can serve the analysis of the said objective(s). We argued that options theory is well suited to address the above mining objectives. Real options analysis recognizes that the value of the capital investment lies not only in the amount of direct revenues that the investment is expected to generate, but also in the future op-

portunities that flexibility creates. An option is an asset that provides its owner the right without a symmetric obligation to make an investment decision under given terms for a period of time into the future ending with an expiration date [Tri95]. If conditions favorable to investing arise, the owner can exercise the option by investing the exercise price defined by the option. A *call option* gives the right to acquire an asset of uncertain future value for the strike price [Tri95].

ArchOptions[Bah05; Bah04], a real options based model which values the *growth* options of an architecture relative to some future changes, as a way for understanding the architectural flexibility with respect to changes in requirements. A growth option is a real option to expand with strategic importance [Tri95] and is common in infrastructure-based investments, as it is the case with software architectures. Since the future changes are generally unanticipated, the value of the growth options lies in the enhanced flexibility of the architecture to cope with uncertainty. ArchOptions builds on a simple and intuitive analogy with Black and Scholes [1973]. In ArchOptions, the flexibility of the middleware induced-architecture in coping with changes in load has a value in the form of growth options. This value is strategic in essence, uncertain as the demand on the future changes are uncertain, and may not be immediate. The added value may take the form of (i) accumulated savings through coping with the change without "breaking" the architecture, mostly these are changes in non-functional requirements; (ii) extending the range of services while leaving the architecture intact; and (iii) the ability to respond to competitive forces and changing market conditions that may pose higher Quality of Service (QoS) requirements, such as the demands for higher availability, scalability, etc.

Choosing a particular middleware to induce the architecture of the software system can be seen as an investment to purchase flexibility in the induced software architecture. The ranges, in which the load changes, influence the choice. A "wise" selection is seen as an investment to buy flexibility, which could be valued as future *growth options* [Tri96] on the architecture of the software system. These *options* enhance the upside potentials of the structure when the load change; they differ from one middleware to another. That is, S_1 is said to be more accommodating to the change than S_0 when S_1 holds more growth options than S_0. For a valuation point of view *p*, we focus the analysis on the calls of the ArchOptions model for valuing the growth options, as given in (1) accounting for both the expected value and exercise cost to accommodate future requirements i_i, for $i \leq n$. Valuing the expectation E of expression *(1)* uses the assumptions of Black and Scholes[Bla73] and detailed in previous work[Bah05; Bah04].

$$\sum_{i=1\ldots n} E\,[\max\,(x_i V_p - C_{eip}, 0)] \qquad (1)$$

The payoff of the constructed call option gives an indication of how valuable the flexibility of an architecture is, when enduring some likely changes in requirements. The selection has to be guided by the expected payoff in $(\sum_{i=1\ldots n} E\,[\max\,(x_i V_p - C_{eip}, 0])_{S1}$ relative to that of $(\sum_{i=1\ldots n} E\,[\max\,(x_i V_p - C_{eip}, 0])_{S0}$. That is, if $(-I_e + \sum_{i=1\ldots n} E\,[\max\,(x_i Vp - C_{eip}, 0)]_{S1} > \sum_{i=1\ldots n} E\,[\max\,(x_i V_p - C_{eip}, 0)]_{S0})$ for some likely changes, then it is worth investing in M_1, as the investment in M_1 is likely to generate more growth options for S_1 than for S_0 and relative to the p valuation point of view. If $(E\,[\max\,(x_k V_p - C_{epk}, 0)])_{S1}=0)$, then M_1 is not likely to payoff, relative to M_0, as the flexibility of the architecture to the change is not likely to add a value for S_1 on *p*, if the change need to be exercised. Two interpretations might be possible: (i) the architecture is overly flexible in the sense that its response to the change(s) has not "pulled" the options relative to *p*. This implies that the embedded flexibility (or the resources invested in implementing flexibility- if any) are wasted and unutilized to reveal the options relative to the changes and relative to *p* (ii) the other case is when the architecture is inflexible relative to the change. This is when the cost of accommodating the change on S_1 is much more than the cumulative expected value of the architecture responsiveness to the change.

Developing the mining tools for extracting patterns serving the chosen economics model(s). Options valuation using Black and Scholes[1973] techniques determine the value of an asset in question in span of the market value using a correlated *twin asset* [Tri95]. The twin asset is an asset that has the same risks as the asset in question will have when the investment has been completed [Sch00]. To understand the behavior of the asset in question, we can use a twin asset, also referred to as a replicated portfolio. The assumption is that under similar conditions the twin asset and the asset in question are interchangeable for all practical purposes and should be worth the same.

Throughput, a scalability measure, expresses the amount of work performed by the system under test during a unit of time. This criterion is based on the observation that for a fixed system with a given throughput (e.g., a single host), there is an inverse relationship between the response time and the number of clients. In other words, the more clients submitting requests, the longer are the delays. A well-known throughput metric is the Total Operations Per Second (TOPS) completed during the measurement interval, referred to as TOPS [http://www.spec.org/]. TOPS is composed of the total number of business transactions completed in the customer domain, added to the total number of work orders completed in the manufacturing domain, normalized per second.

We have mined relevant performance benchmarks, published in (http://www.spec.org/) to understand how the architecture of the system may behave once induced with either J2EE or CORBA with respect to throughput. We appealed to the use of published benchmarks, for the following reasons: First, the system of the given architecture need not be implemented during the evaluation. Thus, performance measures may not be available. Second, we argue that using published benchmarks mimics the concept of the twin asset for we are relying on historical information (though not traded in span of the market, but still hold market information) which shows possible variations in performance in connection to change in load and relative to the candidate implementations. Third, these benchmarks often hint that the throughput is dependent on and can be estimated from the middle-tier "processing power" of the architecture. The advantage of this approach is that the published benchmarks could reveal risks of the operating environment on the choice. Benchmarks are revealing on

the performance dimension because, for example, if multiple benchmarks are conducted with a suitable mix of relevant factors, it may be possible to obtain a set of basic scalability results that can be used for estimating the throughput of possible configurations of the architecture. Depending on the benchmarking algorithm, the relevant scalability factors can be, for example, the number of objects, the number of clients, or the number of nodes in the system etc. supported in response to growing load. A major problem in comparing benchmark results, however, is that different hardware platforms and configurations (e.g., memory, disk drives etc) often produce different results making the comparisons difficult. Fourth, vendors often try many different ways to optimize performance, including adding cache memory and putting cache buffers on disk arrays. This can give a spectrum of worst and best scenarios that could mimics fluctuation, which is a volatility measure, of the option approach.

Analyzing and interpreting the derived patterns. Figure 1 shows the likely throughput trend that the J2EE-induced architecture may exhibit relative to the CORBA-induced one, upon varying the TOPS and the number of hosts. For the J2EE-induced architecture, we provide throughput estimations for two possible implementations: one with JBoss and the other with WLS. For the CORBA-induced architecture, we provide estimates upon the use of JacORB to induce the architecture. Table 1 depicts the upper limit of TOPS supported per host for each of WLS, JBOSS, JacORB induced architectures for 1 to 4 hosts. Figure 2 shows the likely cost-trend upon inducing the Duke's bank architecture with J2EE (using either WLS or JBOSS) and with CORBA (using JacORB). The likely cost is plotted against the number of hosts (1 to 4). The cost refers to the lifecycle cost of the System Under Test (SUT). The cost includes Application Servers/Containers, Database Servers, network connections, etc. Assuming, for example, a five-year lifecycle, cost would include all hardware (purchase price), software including license charges, and hardware maintenance. For the CORBA version, it assumed that the investment incurs an upfront cost to the development of the replication mechanism to support fault-tolerance and load-balancing services for high load scenarios [Bah05]. For the J2EE version of WLS, a license cost is incurred per host.

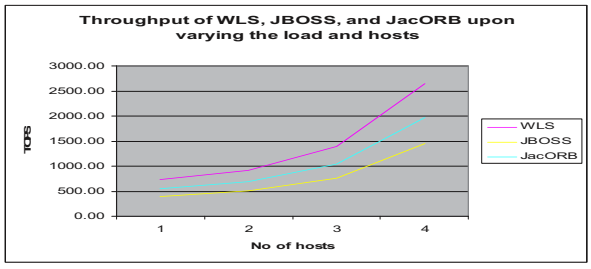


Figure 1. TOPS/host for each of WLS, JBOSS, JacORB (1- 4 hosts)

In [bah05], we have seen that the structural analysis is in favor of the J2EE-induced architecture, the throughput analysis may reveal a different trend upon scaling up each version. From the throughput valuation point of view, Figure 2 shows that when the Duke's architecture will be induced with JBOSS, a J2EE implementation, the system is likely to be slower than that of the JacORB one. This is because JBOSS uses reflection [http://www.jboss.org]. This also implies that there are some chances for the JBOSS-induced architecture to require more hardware for addressing this deficiency. When inducing the Duke's architecture with WLS, another J2EE implementation, the system is very likely to be faster than that of the JacORB implementation. WLS, however, comes with significant licenses costs; this cost grows with the number of hosts, as the load increases. Coining the TOPS with their associated costs, Figure 1, Figure 2 and Table 1, hint that there might be a case for JacORB in certain throughput range. Moreover, once the services for realizing scalability (e.g., the fault-tolerance and load balancing service) are implemented, the cost is incurred once and amortized across the hosts.

Table 1. Upper limit of TOPS/host for WLS, JBOSS, JacORB

Hosts	WLS	JBOSS	JacORB
1	732.00	400.26	546.80
2	918.36	502.16	686.01
3	1395.44	763.03	1042.39
4	2640.96	1444.08	1972.79

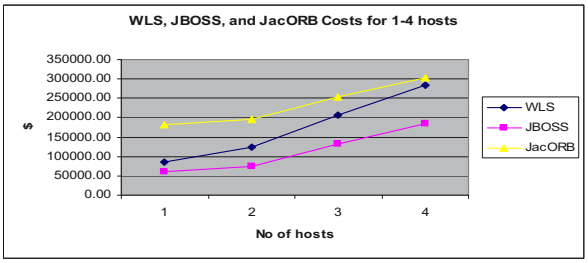


Figure 2. Likely cost-trend upon inducing the Duke's bank architecture with J2EE-(WLS/JBOSS) and with CORBA (JacORB)

Modeling and computation. The case of valuing throughput is appealing to ArchOptions for the following major reasons: First, there is cone of uncertainty associated with the growing load and consequently in the value added as result of our choice. Moreover, the TOPS are of straightforward contribution to value. That is, the more operations are completed per second, the more value is added to the enterprise. However, TOPS incur a price upon executing the operations. The price again is dependent on several factors such as the number of hosts, the hardware, the license cost, and any additional costs that are necessary for making the middleware adaptable to the growing load. In the context of the Duke's Bank, the TOPS range is often uncertain as it is dependent on the customers' behavior at a time. The *uncertainty* in the likely range (i.e., TOPS), the associated costs for executing the TOPS, and the "fluctuation" in the value added as a result make the case very appealing to the use of ArchOptions. Below, we estimate the parameters for computing throughput, Pthro using ArchOptions to address the set mining objectives.

Estimating ($C_{eiPthro}$). TOPS denotes the Total Operations completed per Second. For simplicity of explanation, let us assume that the system of the induced architecture needs to

scale up to support an additional operation per unit-time. An additional operation buys an architectural potential paying an exercise price. In terms of throughput, architectural potential is a performance measure. Hence, what an extra operation pays, if materializes, is a bandwidth for performing that operation. Inducing the Duke's bank with either J2EE or CORBA provide different bandwidth capabilities for performing the operation at different prices. If the implementation of either happens to hold embedded growth options in supporting the extra operation, then the operation is said to pay an exercise price to buy options on the architecture. For the exercise price, we use a well-known normalization factor, which is the *price/performance* [http://www.spec.org/jAppServer2005/] (i.e., the lifecycle cost of the System Under Test (SUT) as configured for the benchmark divided by the throughput). This is provided in the data mined. As an example, assuming five-year lifecycle, the cost would include all hardware (purchase price), software including license charges, and hardware/software maintenance. If the total price is $5,734,417 and the reported throughput is 105.12 TOPS, then the price/performance is $54,551.16/TOPS.

Estimating volatility (σ_{Pthro}). Volatility represents uncertainty attributed to the likely growing of load. For some computation, we abide to the real options principles in computing volatility: we use the standard deviation of x_iV_{Pthro}s due supporting extra operations for a range of load at a host (as the range is said to be revealing to the fluctuation in the value). For other computations, we use modeling estimates for volatility, representing uncertainty to demonstrate how volatility influences the choice and as a way to answer the mining objectives.

Estimating (x_iV_{Pthro}). For simplicity, we estimate x_iV_{Pthro} relevant to the business domain. For every completed online operation, Duke's would not have to serve a customer in person at a branch; the Duke's savings are in the manual-effort for not serving clients at a branch.

Exercise time (t_{Pthro}) and free risk interest rate(r_{Pthro}). As a simulation assumption, we set the exercise time to one year, assuming that the Duke's Bank needs to accommodate the change in one-year time. We set the free risk interest rate to zero (i.e., assuming that the value of money today is the same as that in one year's time).

Results interpretations and analysis. Now, we answer and reflect on the mining objectives to demonstrate usefulness of EDSM. We complement the observed patterns with options computation to inform the problem of tradeoff analyses and decision making in selecting a candidate middleware to induce an architecture, relative to P_{thro}. The likely change in load is the major source of uncertainty that faces Duke's Bank. To address uncertainty and provide better insights on value creation, we have appealed to the use of real options theory. Let us have a close look at the impact of the volatility parameter, which is an expression of uncertainty to address the mining objectives: volatility estimates the "cone of uncertainty" in the future value of the asset, rooted as its current value and extending over time as a function of volatility. As volatility increases, total uncertainty around the benefits also increases. The more TOPS a host is likely to support, the more likely that the actual benefits to "wander" up and down and deviate from the expected present value if the load grows. Let us assume that the present load is in the range of 30- 50 TOPS. Based on the mined data, 30-50 TOPS could be easily addressed by one host using either M_0 (JacORB) or M_1 (Jboss or WLS). For such a low throughput requirements, inducing the architecture with M_0 may appear to be more attractive as when compared to inducing the architecture with M_1 (using either JBOSS or WLS). This is because M_1 incurs license costs for WLS. Moreover, looking at S_1 when induced with JBOSS, S_1 is likely to be in magnitude slower than S_0 as when induced with JacORB due to its use of reflection. This means that S_1 (JBOSS) will support fewer TOPS and consequently will create less value added per second as when compared to S_0. For such low load, the fault-tolerance and load-balancing services need not be implemented on S_0 [Bah 05]. If options analysis is not used, M_0 will be a no-brain choice for inducing the Duke's Bank architecture. Though inducing the architecture S_1 with M_1 (using WLS) appears less attractive than M_0 (JacORB), S_1 may still carry embedded growth options which will only materialize if the load grows. If we use a Present Value (PV), the computation will based on the benefits of supporting the TOPS less their costs (i.e., the computation does not account for uncertainty). The resulted valuation will compute the present value as realized and ignore the growth options. In other words, inducing the architecture with WLS if undertaken, PV would hint that S_1 would destroy value rather than create it. That is, *Value* S_1 = PV. However, *Value* S_1 is actually *Value* S_1 = PV + *Opt*. That is, M_1 carry embedded growth options, *Opt*. The *Opt*, if left unexercised, are ignored by the non-options analysis. Hence, *Value* for S_1 is then said to be underestimated. As a result, S_0 may look more attractive (Table 2). The Present Value calculation of Table 2 shows that S_1 is the least attractive for this range of load. The computation is based on the benefits of supporting 100 TOPS less their costs. However, the computation ignores the growth options on S_1 in supporting additional 632 TOPS using the first host. Similarly, PV systematically undervalues the growth potential of S_1 (JBOSS) and S_0 (JacORB) in respectively supporting 300.26 TOPS and 446.26 TOPS. That is, PV ignores the flexibility value of S_1 and S_0 in responding to the growing load at host 1.

Table 2. Illustration PV per second ($) for low throughput (100 TOPS)

100 TOPS	Max TOPS	$C_{eiPThro}$	X_{VPThro}	PV	Value Ignored (TOPS)
S_1(WLS)	732.00	853.11	12.63	-840.48	-632
S_1(JBOSS)	400.26	603.11	12.63	-590.48	-300.26
S_0(JacORB)	546.80	603.11	12.63	-590.48	-446.80

It is a fact that PV does not work well for projects with future decisions that depend on how uncertainty resolves. Though they can be used to evaluate the operational benefits in a stable environment with well-understood and measurable costs and benefits, they have little to offer when capturing additional value due to flexibility under uncertainty, such as strategic opportunities and the ability to respond to changing

conditions. Using PV, S_1, when induced with WLS, reports negative values upon inducing the architecture with WLS for this range of load. However, the situation indicates that these results underestimate the value of S_1, as S_1 can better respond to uncertainty, where the load is likely to grow over 100 TOPS. In Table 3, we turn to ArchOptions to capture the growth options on S_1 and S_0. The volatility parameter is an expression of the range of "benefits" at a host. For S_1 (WLS): the benefits could "wander" from zero (i.e., idle state with no operations executing at a second) to the benefits derived from full utilization of capacity (i.e., in the support of 732 TOPS). That is, the volatility of 66% for S_1 (WLS) indicates that the benefits of executing the TOPS is in the range of $0(idle) to $92.42(full utilization) per second on host 1. Similarly, for S_0 (JacORB): the 45% volatility for S_0 (JacORB) indicates that the benefits of executing the TOPS are in the range of $0(idle) to $69.04 (full utilization) per second on host 1. As for the options on S_1(WLS), S_1 has "pulled" the options on one host for this range of load. This is because we have accounted for the possible fluctuation in the derived values from supporting the TOPS. Considering such "fluctuation" provides us with better insights on the architectural potential of S_1 in support of this likely change in load. Table 3 suggests S_1 has reported a value added of $0.017 on 1 host.

Table 3. Illustration options per second ($) very low throughput scenario (100 TOPS)

100 TOPS	$C_{eiPThro}$	$X_{iVP\text{-}Thro}$	σ_{Pthro}	Options	Actual Value (TOPS)
S_1(WLS)	853.11	92.42	66%	0.01700	100 + 632
S_1(JBOSS)	603.11	50.53	35%	0+	100 + 300.26
S_0(JacORB)	603.11	69.04	49%	0.00001	100 + 446.80

4. Related Work

Mining Software Repositories (MSR) [MSR 1-5] is a growing community in Software Engineering. [http://msr.uwaterloo.ca/] provides excellent up-to-date reference to MSRs. These contributions, however, are essentially technical endeavor with no attention paid to the economics context: software repositories are often mined and analyzed ignoring the link between technical properties, economics, and value creation under a given circumstances. Our contribution is novel in addressing this gap. [EDSR 1-8] community is interested in linking technical engineering concepts to economics and value creation. No contribution has been reported on EDSM, except for [Bah07] bridging the gap between these two communities.

5. Conclusion

We have highlighted a scenario for realizing EDSM through an example. The example describes how software repositories could be mined to value the ranges in which a given software architecture can scale to support likely changes in load. The exposed arguments provide an example of the invaluable insights that the analyst might benefit from upon complementing the mined data with economics computation. Such analysis has the promise to provide the software analyst with a powerful tool for predicting cost/value information for developing and evolving dependable software and understanding the economics ramification of the change on the system and its design artifacts (e.g., architectures); and informing design trade-offs. The objective is to provide insights into investment decisions related to the development and evolution of software systems and assisting in resource planning and utilization. Ongoing work includes designing an automated infrastructure and tools support. Effort includes designing a semi-automated support for executing the EDSM process, deriving interesting patterns, facilitating the computation, visualizing the results, assisting in interpretations, and supporting sensitivity analyses. Interestingly, MSR [1-5] drew the attention to a new challenge faced by empirical studies: whereas previous studies suffered from lack of data, current studies face challenges dealing with enormous amounts of freely available data from easily accessible repositories online such as forums, code, and bug reports repositories. Though this fact may have implications on the quality of the mined data and the resulted analysis, this could also hint to opportunities for EDSM, where existing knowledge could provide insights into investment decisions related to development and evolution of systems. This could, for example, be based on analogies and similar to the way we have "mimicked" the concept of twin asset.

6. References

[Bah04] Bahsoon, R. and Emmerich, W.: Evaluating Architectural Stability with Real Options Theory. In: Proc. of the 20th IEEE Int. Conf. on Software Maintenance (2004)

[Bah05] Bahsoon, R., Emmerich, W., and Macke, J.: Using ArchOptions to Select Stable Middleware-Induced Architectures. In: IEE Proceedings Software, Special issue on Relating Requirements to Architectures, IEE Press 152(4) (2005) 176-186

[Bah07] Bahsoon, R. and Emmerich, W.: Economics-Driven Software Mining. In: Proc. of the ICSE 2007 workshop on Economics of Software and Computation.

[Bah08] Bahsoon, R. and Emmerich, W.: (2008) An Economics-Driven Approach for Valuing Scalability in Distributed Architectures. In Proc. of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA 2008).

[Bla73] Black, F., and Scholes, M.: The Pricing of Options and Corporate Liabilities. Journal of Political Economy. U. of Chicago Press (1973) 637-654

[Boe00] Boehm, B., and Sullivan, K. J.: Software Economics: A Roadmap. In: A. Finkelstein (ed.): The Future of Software Engineering. ACM Press (2000) 320-343

[DiN99] Di Nitto, E., and Rosenblum, D.: Exploiting ADLs to Specify Architectural Styles Induced by Middleware Infrastructures. In: Proceedings of the 21st Int. Conference on Software Engineering, ACM Press (1999) 13-22

[EDSR 1-8] EDSER 1-8: Proceedings of the Workshops on Economics-Driven Software Engineering Research: In conj. with the 21st through 28th International Conference on Software Engineering (1999 - 2006)

[Emm00] Emmerich, W.: Software Engineering and Middleware: A Road Map. In: A. Finkelstein (ed.), Future of Software Engineering, ACM Press (2000) 117-129

[Min99] Mining Software Engineering Data: A Survey

[MSR1-5] MSR 1-5: Proceedings of the ICSE Workshops on Mining Software Repositories, In conjunction with ICSE 2004- 2008.

[OMG00] Object Management Group: The Common Object Request Broker: Architecture and Specification, 2.4 ed., OMG (2000)

[Sul99] Sullivan, K. J.: Chalasani, P., Jha, S., and Sazawal, V.: Software Design as an Investment Activity: A Real Options Perspective. Real Options and Business Strategy: Applications to Decision-Making. In: Trigeorgis L. (ed.) Risk Books (1999) 215-260

[Sun] Sun Microsystems Inc.: Duke's bank application, http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Ebank.html

[Tri95] Trigeorgis, L.: Real options in Capital Investment: Models, Strategies, and Appications. Praeger Westport, London (1995)

VP: AN EFFICIENT ALGORITHM FOR FREQUENT ITEMSET MINING

Qin Ding[1], Wen Shen Huang[2]

[1] *Department of Computer Science, East Carolina University, Greenville, NC 27858, USA*
dingq@ecu.edu

[2] *Department of Computer Science, Pennsylvania State University - Harrisburg, Middletown, PA 17057, USA*

Abstract *– Frequent itemset mining is one of the important problems in data mining. The task is to discover frequent occurring patterns in large databases. Many algorithms have been proposed for frequent itemset mining, such as the Aprioir and FP-growth algorithms. In this paper, we propose the Virtual Partition Algorithm (VP) for frequent itemset mining. The VP algorithm treats a database as small partitions (projected databases) and mines frequent itemsets from each partition. The search space is a lexicographical tree. A node of the tree, denoted P_F, represents a distinct partition of the database, where all transactions in the partition contain the frequent itemset F. The VP algorithm uses strategies such as virtual partition, transaction assignment, transaction reassignment, and pruning techniques to improve the efficiency and memory utilization. We compared our algorithm with other frequent itemset mining algorithms and the experimental results show that VP is efficient for frequent itemset mining on both sparse and dense databases.*

I. INTRODUCTION

Frequent itemset mining is an important problem in data mining. The task is to identify items that are frequently occurring together in databases. Frequent itemset mining is a sub-task and also the major task in association rule mining [3, 4]. Association rule mining aims to discover relationships (such as associations and correlations) among different items in database. An association rule is in the format of "X => Y" where X and Y are set of items, called itemsets. This rule indicates that the presence of X implies a strong possibility of the presence of Y. Two measurements, support and confidence, are used to indicate the strength of an association rule. The support of the rule X => Y is the fraction of the transactions containing both X and Y. The confidence of X => Y is the fraction of transactions containing X which also contain Y. To discover all the association rules in a database, most algorithms first identify frequent itemsets, which are itemsets with support above a minimum support threshold.

A number of algorithms have been proposed for frequent itemset mining, one of which is Apriori algorithm [4]. Apriori is a step-wise algorithm using candidate generation. Candidate itemsets are potentially frequent itemsets. Apriori iteratively generates candidate (k+1)-itemsets from the complete frequent k-itemsets and then tests the candidates against the database. It utilizes the Apriori property that if an itemset is frequent then all of its subsets must also be frequent to reduce the size of the candidate set significantly. However, when the number of frequent itemsets is large and/or long frequent itemsets exist, it still suffers from generating a considerable number of candidates and from performing tedious database scans and pattern matching to determine the support of each candidate.

Apriori-based algorithms are not efficient to mine frequent itemsets from dense databases. A dense database has any or all of the following properties: many frequently occurring items, strong correlations between several items, and many items in each transaction. On the other hand, in sparse databases, items do not occur frequently and in general frequent itemsets are relatively short. In order to deal with dense databases or long frequent itemsets, strategies trying to avoid the candidate generation have been proposed, among which is the FP-growth algorithm [9]. Instead of generating candidates, FP-growth utilizes a special tree structure, FP-tree, to maintain the necessary information, and then mines frequent itemsets in a bottom-up manner along the tree. FP-growth is efficient as it avoids the candidate generation and it needs only two database scans. Nevertheless, FP-growth is known as a very memory-consuming approach because it requires significant memory space when recursively constructing conditional FP-trees and it is especially true if the database is huge and sparse.

Generally speaking, the difficulties of the frequent itemset mining problem are candidate generation, pattern matching, and the requirement of large memory space. In this paper we propose the Virtual Partition (VP) algorithm to solve this problem and to achieve the following goals:

- Deal with different types of databases efficiently
- Mine frequent itemsets without candidate generation
- Avoid pattern matching, i.e., checking candidate itemsets against each transaction
- Reduce database scans and the search space
- Minimize the usage of memory space
- Allow for scalability and reliability

It should be noted that in this paper we assume that the employed representation of the database fits in main

memory. If it is not the case, an approach based on opportunistic projection [10] can be applied, but this is beyond the scope of our work.

The rest of this paper is organized as follows. Section II discusses related work. Section III details the Virtual Partition (VP) algorithm. The experimental results are given in Section IV, and Section V concludes the paper.

II. RELATED WORK

Apriori and FP-growth algorithm represent two categories of approaches used in frequent itemset mining, one using candidate generation and one without. The Apriori algorithm and some Apriori-like algorithms are widely used in mining frequent itemsets. However, if the database size is large and/or the minimum support threshold is low, the large number of candidates plus the cost of database scans will degrade the performance. FP-growth algorithm does not need to generate candidate itemsets. It has been proved to be one of the most efficient algorithms for the frequent itemset mining problem; it is especially true when mining frequent itemsets in dense databases. However, FP-growth is a very memory-consuming algorithm.

Besides Apriori and FP-growth, there are a number of other algorithms for mining frequent itemsets, such as Eclat [18] and Partition [15]. Eclat uses vertical layout of transactions instead of traditional horizontal layout. The partition algorithm divides a large database into a number of smaller databases that can fit in memory and finds local frequent itemsets in each partition and then combine them into global frequent itemsets.

In general, no single approach outperforms others for all cases in frequent itemset mining because of the variety of databases. Different databases have different sizes, densities, and layouts. Therefore, choosing a proper approach is nontrivial. Recently, researchers have put efforts in developing hybrid methods such as H-mine [13], kDCI [11], and Nonordfp [14]. They employ two alternative strategies to deal with sparse and dense databases, and switch adaptively from one to the other according to the information collected during execution phases. However, questions including what should be the appropriate situations that one strategy is more preferable over the other and how to determine when such a strategy switching should happen are still nontrivial.

III. VP: THE VIRTUAL PARTITION ALGORITHM

A. Algorithm Overview

In order to mine frequent itemsets without candidate generation and pattern matching, we first propose the Tree Generation algorithm (TG). The TG algorithm mines the complete set of frequent itemsets from a database by constructing a Prefix-Pattern Tree (PP-Tree). A node of the tree is denoted P_F , where F is a frequent itemset and we call it a prefix-pattern since all frequent itemsets that can be recursively mined from P_F must have F as their prefix. For example, the frequent itemsets that can be recursively mined from $P_{\{1, 3\}}$ must have itemset {1, 3} as their prefix. Each prefix of F is the prefix-pattern of a corresponding ascendant node of P_F. For example, $P_{\{2\}}$ and $P_{\{2, 3\}}$ are the ascendant nodes of $P_{\{2, 3, 4\}}$. Moreover, $P_{F'}$ is one of the ascendant nodes of P_F if and only if $P_{F'}$ is a prefix of F.

TreeProjection [1], another tree-based algorithm, mines frequent itemsets by projecting transactions of a given database in a divide-and-conquer manner. Instead of projecting transactions, TG constructs a PP-Tree by dividing a large database into smaller portions and then recursively mines frequent itemsets from each portion in a depth-first manner. Such a portion is called a partition of the database, or a partition. Each node of the tree corresponds to a distinct partition. When no further nodes can be generated, the mining task completes, and each node of the tree indicates a distinct frequent itemset. The prerequisite of this algorithm is that an ascending or lexicographical ordering exists among the items in the database.

To overcome difficulties of TG, including transaction replication and physical partition, we propose the Virtual Partition algorithm (VP). VP is based on TG, and we use the term "virtual" because VP mines frequent itemsets without physically partitioning a database. VP constructs and utilizes an Item Linked-list Structure (ILS) to simulate behaviors including node generation, transaction assignment, and transaction reassignment involved in TG.

B. Search Space

The search space of TG is considered a PP-Tree. We define ζ to be the last item in prefix-pattern F. For example, if F = {2, 3, 4}, ζ = 4. Each node P_F of the tree is labeled by ζ and corresponds to a distinct partition of a database, where F is the set of ζ on the path from the root to P_F itself. For example, in node $P_{\{\varnothing, I1, I2, \ldots, Ii, \ldots, \zeta\}}$, F = {$\varnothing$, I_1, I_2, …, I_i, …, ζ}, and the path $P_{\{\varnothing\}} \rightarrow P_{\{\varnothing, I1\}} \rightarrow P_{\{\varnothing, I1, I2\}} \ldots \rightarrow P_{\{\varnothing, I1, I2, \ldots, Ii\}} \ldots \rightarrow P_{\{\varnothing, I1, I2, \ldots, \zeta\}}$ must exist. The ith node is at $Level_i$, the ith level of the PP-Tree. For simplicity, we ignore $\varnothing$ since the root represents the empty set; therefore, the cardinality of F is the level of the tree.

Given a database D comprising I: {1, 2, 3, 4, 5}, the maximum PP-Tree is shown in Fig. 1. This tree is a lexicographical tree that consists of I except for the last item 5. The last item of I can be ignored because no superset of such item exists. A node P_F of the tree is the node from which the complete set of frequent itemsets containing F can be recursively mined. For example, the frequent itemsets containing itemset {2, 3, 4}, if existing, can be recursively mined from $P_{\{2, 3, 4\}}$.

If frequent k-itemsets exist, the complete set of frequent k-itemsets can only be mined from nodes at $Level_{k-1}$. All transactions in node P_F at $Level_k$ contain the same frequent k-itemset F. The frequent supersets that contain F can only be mined from P_F and its descendant nodes. Since any frequent itemset is a subset of I, the maximum level of the prefix tree is smaller than the cardinality of I. For example, the maximum frequent itemsets is {1, 2, 3, 4, 5}, and it can

be mined from $P_{\{1, 2, 3, 4\}}$ at $Level_4$; thus, the maximum level of the PP-Tree is 4.

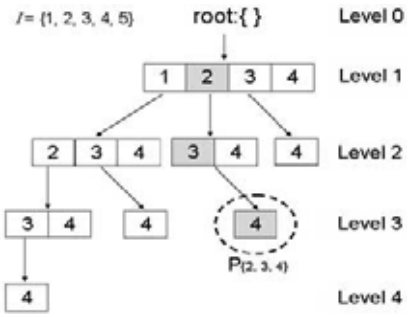


Fig. 1. The Maximum PP-tree

C. Algorithm Details

The TG algorithm is primarily based on the breadth-first strategy for node generation, combined with the depth-first strategy for database partition and transaction reassignment. A PP-Tree is constructed in a top-down fashion by starting from the root node and successively generating nodes until no further nodes can be generated.

The mining process consists of constructing a PP-tree, where each node of the tree corresponds to a distinct frequent itemset of a database. A node PF is generated if and only if frequent itemset F is mined. TG eventually constructs a PP-Tree once a mining task completes. The finally constructed PP-Tree depends on the value of the minimum support threshold ξ, and this tree is part of the maximum PP-Tree, i.e, only some nodes of the maximum PP-Tree are generated during mining.

Fig. 2 illustrates the Tree Generation algorithm. To mine the transactional database D, TG scans D to determine the frequent 1-itemsets R. Next, according to R, it generates nodes. Nodes are initially active. A node is inactive if no further frequent itemsets can be mined from it and its descendants, otherwise it is active. Assuming R = $\{I_1, I_2, \ldots, I_n\}$, the newly generated nodes are $P_{I_1}, P_{I_2}, \ldots, P_{I_{n-1}}$, and P_{I_n} is not generated. After the node generation, TG partitions D by assigning transactions onto the newly generated nodes at $Level_1$.

For each node P_F, if some frequent itemsets exist, TG calls the recursive procedure SEARCH to mine the complete set of frequent itemsets containing F. Once no further frequent itemsets can be mined in P_F and its descendant nodes, P_F becomes inactive, and those transactions in P_F must be reassigned onto other active sibling nodes because some frequent itemsets that do not contain F are still un-mined and can be mined in such active nodes. In other words, in order to reflect the actual support of those un-mined frequent itemsets, a transaction T containing any of such frequent itemsets in P_F needs to be eventually reassigned onto some corresponding active nodes.

The number of transactions that need to be assigned and reassigned significantly decreases as the mining process proceeds. In addition, TG counts frequent items locally, avoids pattern matching, and maintains only a portion of the database during mining.

Input: A transactional database D and the minimum support threshold ξ
Output: *Result*: the complete set of frequent itemsets in D.

```
1)  Let Result, R = ∅;
2)  Scan D to do counting; R ← frequent 1-itemsets;
3)  if R ≠ ∅ do {
4)      Result ← R; Set R = ∅;
5)      Generate active nodes at Level_1;
6)      Scan D again to assign transactions from D onto nodes at Level_1;
7)      for each node P_F at Level_1 do { //F is the prefix-pattern of the node
8)          Let k = 2;
9)          SEARCH(P_F, k); //Mine the complete set of frequent itemsets containing F
10)         Set P inactive;
11)         Reassign transactions from P onto active sibling nodes;
12)     }
13) }

14) Procedure SEARCH(node P_F, int k) {
15)     Scan P_F to do counting; R ← frequent 1-itemsets;
16)     if R ≠ ∅ do {
17)         Result ← GetResult(F, R); Set R = ∅;
18)         Generate its active child nodes at Level_k;
19)         Scan P_F again to assign transactions from P_F onto its child nodes at Level_k;
20)         for each child node of P_F, P_F', do {
21)             SEARCH(P_F', k + 1);
22)             Set P_F' inactive;
23)             Reassign transactions from P_F' onto other active sibling nodes;
24)         }
25)     }
26) }

27) Procedure GetResult(prefix pattern F, frequent 1-itemsets R) {
28)     Let R̄ = ∅;
29)     for each item α in R do {
30)         R̄ ← F ∪ α; //each F ∪ α is a new frequent itemset
31)     }
32)     return R̄;//R̄ is the complete set of frequent (k + 1)-itemsets containing F,
                //where k is the cardinality of F
33) }
```

Fig. 2. The Tree Generation Algorithm

IV. EXPERIMENTAL RESULTS

We implemented VP in C++. Our experiments were conducted on a 2.4GHz Intel Pentium IV processor with 512MB main memory running Linux Debian.

The algorithms were tested on the datasets shown in Table 1, available on the FIMI'03 frequent itemset mining benchmark website. The two synthetic datasets, T10I4D100K and T40I10D100K, were generated using the generator2 from the IBM Almaden Quest research group that simulates the buying behavior of customers in retail business. The parameters for generating a synthetic dataset include the number of transactions D (in thousands), the average transaction size T, and the average length I of so-called maximal potentially large itemsets. For example, dataset "T10I4D100K" has an average transaction size of 10, an average size of the maximal potentially frequent itemsets of 4, and 100,000 generated transactions. Clearly the T40I10D100K dataset is denser than the T10I4D100K dataset. The Mushroom dataset contains characteristics from different species of mushrooms. The Chess dataset contains different game configurations. The Retail dataset is the retail market basket data set supplied by an anonymous Belgian retail supermarket store.

Table 1. Test Datasets

Dataset	# of Items	Avg. Length	# of Transactions
T10I4D100K	1000	10.0	100,000
T40I10D100K	1000	39.6	100,000
Chess	75	37	3,169
Mushroom	119	23	8,124
Retail	16,469	10.3	88,162

We examined the execution time and the memory utilization in our tests. The execution time is in seconds and excludes the preprocessing for each algorithm, which includes reading databases from files and constructing data structures, or tries, representing such databases before mining. It also excludes the time needed to print the resulting itemsets. These excluded processes together usually take a few seconds. In the experiments, we did not record the execution time of a mining task if its runtime exceeded 10 minutes, and we recorded a task as a "timeout" if it was unable to complete mining within 10 minutes. To measure the memory utilization of a process, we monitored the value of VIRT, associated with such process, in top output, where "top" is a traditional Unix memory management tool and VIRT stands for the virtual size of a process. The domain of VIRT includes the sum of memory a process is actually using, memory it has mapped into itself, and memory shared with other processes. In short, VIRT represents the amount of memory that a process can access at the present moment. The unit of VIRT is in KB when the memory utilization is less than 100MB, and it converts to MB when the memory utilization exceeds 100MB. The approximate memory utilization is suitable to our test because we can observe the trend and variation of the memory utilization as the minimum support threshold decreases. In the experiments, we terminated a task if its memory utilization was large enough to potentially crash the system, and we recorded a task as "aborted" if its memory utilization was too large.

We compared VP with the following frequent itemset mining algorithms: APRIORI [5], COFI [12], and CT-PRO [16]. Categorically speaking, VP and APRIORI are counting-based algorithms, and the other two algorithms are based on FP-growth. APRIORI is one of the most efficient implementations for Apriori-based algorithms. COFI introduces a simple and non-recursive mining process to replace the memory-based FP-tree. CT-PRO uses a more compact data structure, the Compressed FP-Tree (CFP-Tree), to non-recursively mine frequent itemsets in a bottom-up fashion. CT-PRO performs better than OpportuneProject [10], FP-Growth [9], Apriori [4], LCM [17], and kDCI [11], where LCM and kCDI are known as the two best algorithms in FIMI 2003 repository.

The minimum support greatly affects the runtime. In a dense dataset, items are strongly correlated; therefore, given the same support value, the number of frequent itemsets that can be mined from dense datasets is much larger than the number of frequent itemsets that can be mined from sparse datasets. Mining more frequent itemsets requires more execution time. Accordingly, in our experiments, we used very small support values for T10I4D100K, T40I10D100K, and Retail datasets. In contrast, the support values used for Chess and Mushroom were very high. Moreover, in order to examine the efficiency, reliability, and scalability of the four algorithms, we used extremely small support values for each experiment so that we can test which algorithms can complete mining tasks under those situations.

Tables 2, 3, 4, 5, and 6 summarize the runtimes (in seconds) of the four algorithms on these datasets along with the total number of frequent itemsets (# of FISets) and the length of the maximum frequent itemsets (MaxLen of FISets) for each given minimum support threshold value (Minsup).

Table 2. Execution Time (in seconds) for Mining T10I4D100K

Minsup(%)	APRIORI	COFI	VP	CT-PRO	# of FISets	MaxLen of FISets
1.0	1.56	**0.3**	0.34	0.4	386	3
0.5	2.22	0.5	**0.48**	0.5	1,074	5
0.1	4.81	1.6	**1.26**	1.8	27,533	10
0.05	7.31	10.7	**1.56**	10.6	53,386	10
0.025	14.21	95.5	**1.94**	95.5	96,969	11
0.01	85.15	aborted	**3.42**	aborted	411,366	13
0.005	aborted		**7.59**		1,923,260	13

Table 3. Execution Time (in seconds) for Mining T40D10I100K

Minsup(%)	APRIORI	COFI	VP	CT-PRO	# of FISets	MaxLen of FISets
5	4.88	2.3	**1.97**	4.43	317	2
2.5	6.06	5.0	**4.41**	7.41	1,222	2
1.75	8.27	45.6	**7.56**	12.79	3,393	3
1.5	12.3	aborted	**9.85**	18.44	6,540	7
1.25	**27.87**		**13.3**	29.74	13,461	11
1.0	46.13		**25.28**	56.7	65,237	13
0.75	202.16		**78.36**	133.51	496,883	18
0.50	343.39		**146.58**	480	1,286,038	18
0.40	464.07		**185.25**	timeout	1,973,396	18
0.30	timeout		**293.87**		5,058,313	20

Table 4. Execution Time (in seconds) for Mining Chess

Minsup(%)	APRIORI	COFI	VP	CT-PRO	# of FISets	MaxLen of FISets
90	0.1	**<0.01**	0.12	0.03	629	7
80	0.22	0.2	1.90	**0.02**	8,228	10
70	2.93	3.7	8.62	**0.05**	48,970	13
65	18.5	16.9	20.47	**0.07**	111,779	13
60	84.59	58.7	41.02	**0.12**	254,945	14
55	189.26	aborted	84.64	**0.22**	574,999	15
50	514.19		156.75	**0.43**	1,272,933	16
40	timeout		598.32	**1.92**	6,472,981	18
30			timeout	**66.93**	37,282,963	20

Table 5. Execution Time (in seconds) for Mining Mushroom

Minsup(%)	APRIORI	COFI	VP	CT-PRO	# of FISets	MaxLen of FISets
30	0.2	**<0.01**	1.37	0.05	2,736	9
20	4.9	0.5	8.59	**0.07**	53,584	15
15	12.67	4.7	12.46	**0.09**	98,576	15
12.5	23.78	9.7	14.49	**0.1**	118,456	15
10	108.5	aborted	34.55	**0.21**	574,514	16
7	232.45		66	**0.41**	1,356,602	16
6	259.42		69.58	**0.47**	1,454,180	16
5	548.66		121.86	**0.96**	3,755,706	17
4	timeout		137.3	**1.12**	4,390,342	17
2			394	18.22	23,596,650	18
1.5			**588.48**	aborted	43,143,418	18

Table 6. Execution Time (in seconds) for Mining Retail

Minsup(%)	APRIORI	COFI	VP	CT-PRO	# of FISets	MaxLen of FISets
1	0.96	**<0.01**	0.19	0.28	160	4
0.5	1.18	**0.1**	0.43	0.5	581	5
0.1	3.4	**0.4**	2.84	1.96	7,713	5
0.05	5.81	**0.8**	2.75	2.91	19,837	7
0.025	8.93	**2.6**	**5.75**	3.91	50,481	8
0.02	12.39	4.2	12.3	**4.19**	67,186	9
0.015	14.65	10.4	24.93	**7.83**	113,119	9
0.0125	17.58	21.2	31.08	**13.78**	155,111	10
0.01	**24.46**	aborted	43.38	aborted	240,852	13

By observation, as the support value declined, the length of maximum frequent itemsets grew evenly and the number of frequent itemsets escalated significantly. When the support value was relatively high, most frequent itemsets were of short lengths and thus the four algorithms had similar performances; on the other hand, most items were frequent when support values became small.

For datasets T10I4D100K and T40I10D100K, APRIORI and VP had good performance. In particular, VP outperformed the other three algorithms when support values were very small. As indicated in Table 2, COFI and CT-PRO had similar performance, and both aborted when the support was 0.01%. When the support declined from 0.01% to 0.005%, the number of frequent itemsets increased approximately 4.68 times, and VP completed the task in only 7.59 seconds while others failed. Compared to T10D4I100K, T40D10I100K is larger and denser. In Table 3, VP was still the best algorithm. In general, VP was competitive for most given support values, and it was even more obvious when the support value was very small. For instance, VP was 2.34 times faster than APRIORI and 3.27 times faster than CT-PRO when the support value was 0.5%, while COFI aborted when the support value was 1.5%.

When mining in very dense datasets such as Chess and Mushroom, the FP-growth-based algorithms outperformed the counting-based algorithms. However, it was true only when available memory was sufficient. CT-PRO utilizes the CFP-tree data structure, known to be very memory efficient for very dense datasets. Accordingly, although both COFI and CT-PRO are FP-growth based algorithms, CT-PRO was very efficient (as shown in Tables 4 and 5) as COFI suffered from inefficient memory utilization.

VP was unable to outperform CT-PRO in dense datasets, but it was reliable in that it never aborted for any given support value. Although VP was recorded timeout in Table 4.4 when the support value was 30%, we found that it can finish the task eventually if we let it continue. Moreover, in Table 5, when the support value was 1.5%, VP completed the task while CT-PRO was not able to do so.

Retail is a very sparse dataset. It consists of 16,469 items and the average length of transactions is short. When mining Retail, most algorithms were efficient, and the runtimes had the same orders of magnitude. The performance of VP was better than that of APRIORI when the support value was larger than or equal to 0.02%. Furthermore, VP was more reliable than COFI and CT-PRO because VP survived when the support value was as small as 0.1%, while COFI and CT-PRO both aborted. We found that VP was not as fast as we expected since the number of items was very large.

VP created header tables in each iteration and the sizes of tables were decided by the size of items. The time for memory allocation became relatively expensive as the computational cost was relatively slight for all algorithms in very sparse datasets. Moreover, each header table needs to be scanned for the node generation. To improve the performance of VP for mining datasets with large set of items, reusing the header table may be a promising strategy.

Tables 7 and 8 illustrated the memory utilization, in megabytes (MB), of the four algorithms for mining T40D10I100K and Retail, respectively. We found that COFI is a very memory demanding algorithm even though [12] declared that it can significantly reduce the candidate generation and avoid recursion. COFI was the most unscalable algorithm among the four since it was very sensitive to the length of maximum frequent itemsets. CT-PRO usually took more memory utilization than others and it was infeasible for mining very large datasets. The memory utilization of APRIORI is based on the size of number of frequent itemsets. For instance, as illustrated in Table 7, when the support value decreased from 0.75% to 0.5%, the memory utilization of APRIORI increased 105.6% while the number of frequent itemsets increased 158.8%. On the other hand, given the same condition, the memory utilization of VP increased only 0.4%.

Table 7. Memory Utilization (in MB) for Mining T10I4D100K

Minsupt(%)	APRIORI	COFI	VP	CT-PRO	# of FISets	MaxLen of FISets
5.0	**15**	74.8	24.6	124	317	2
2.5	**19.5**	99.8	31.3	174	1,222	2
1.75	**21.4**	135	32.8	188	3,993	3
1.5	**22.7**	> 600	33.3	193	6,540	7
1.25	**28.1**	-	33.7	197	13,461	11
1.0	**32.1**	-	34	201	65,237	13
0.75	55	-	**34.3**	208	496,883	18
0.5	113	-	**34.5**	222	1,286,038	18
0.3	309	-	**34.7**	309	5,058,313	20

Table 8. Memory Utilization (in MB) for Mining T40D10I100K

Minsup(%)	APRIORI	COFI	VP	CT-PRO	# of FISets	MaxLen of FISets
0.05	37.4	18.2	**9.3**	150	19,837	7
0.025	76.2	44.9	**10.5**	285	50,481	8
0.02	95.1	50.2	**11.1**	392	67,186	9
0.015	112	62.5	**11.5**	499	113,119	9
0.0125	128	73.3	**11.6**	548	155,111	10
0.01	159	537	**11.8**	> 600	240,852	13
0.0075	190	> 600	**12**	-	480,621	15
0.005	354	-	**12.8**	-	1,506,775	17

During mining, the memory utilization of COFI varied because of the requirement of memory allocation, deallocation, and reallocation. The memory utilization of CT-PRO does not change once the CFP-tree was established, and the memory utilization of APRIORI steadily climbed up because it recorded frequent (k + 1)-itemsets generated from candidate k-itemsets.

VP was insensitive to the length of maximum frequent itemsets and/or the number of frequent itemsets. Similar to CT-PRO, the memory utilization of VP increased slightly once ºA was established where ºA is a vector used to represent a loaded database. During the rest of mining, VP did not generate extra complicated data structures or tries such as hash-tables or FP-trees, except for some smaller header tables. Moreover, the memory utilization of VP was always relatively small for any given dataset and any given Minsup. In fact, the memory utilization of VP was the sum of the size of ºA and the size of header tables, and we can always expect the approximate memory utilization in advance. In short, VP is very scalable and efficient for the

memory utilization and hence it is the most reliable algorithm in our experiment.

V. CONCLUSIONS AND FUTURE WORK

Frequent itemset mining is an important task in data mining. Various algorithms have been proposed for frequent itemset mining. In this paper we proposed an algorithm called "Virtual Partition" (VP) to deal with different kinds of databases in an efficient way to mine frequent itemsets. VP utilizes the ILS structure to simulate the database partition and transaction reassignment involved in the tree projection algorithm. It mines frequent itemsets without candidate generation, avoids actual data moving, and maintains minimum memory utilization. We compared VP with several other algorithms on different datasets, including dense and sparse datasets. Our experimental results are generally quite competitive, although we do not excel in every case. Nonetheless, VP never aborted while other algorithms sometimes failed. We conclude that VP is scalable and reliable for frequent itemset mining problem.

To improve our algorithm in mining very dense databases, we may combine VP with CT-PRO or other algorithms that can mine dense databases more efficiently. Furthermore, instead of using the lexicographical ordering, we may consider sorting items in frequency order for each transaction. This could increase the preprocessing time for sorting the items of each transaction, but the performance gain may be promising.

REFERENCES

[1] R. Agarwal, C. Aggarwal, and V. V. V. Prasad: "A Tree Projection Algorithm for Generation of Frequent Itemsets". In Journal of Parallel and Distributed Computing, Vol. 61, No. 3, 2001, pp. 350-371.

[2] R. Agarwal, C. Aggarwal, and V. V. V. Prasad: "Depth First Generation of Long Patterns". In Proc. of the ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining, 2000, pp. 108-118.

[3] R. Agarwal, T. Imielinski, and A. Swami: "Mining Association Rules between Sets of Items in Very Large Databases". In Proc. of the ACM SIGMOD Int'l Conference on Management of Data, 1993, pp. 207-216.

[4] R. Agarwal and R. Srikant: "Fast Algorithms for Mining Association Rules". In Proc. of the Int'l Conference on Very Large Databases (VLDB), 1994, pp. 487-499.

[5] F. Bodon: "A Fast Apriori Implementation". In Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, 2003.

[6] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur: "Dynamic Itemset Counting and Implication Rules for Market Basket Data". In Proc. of the ACM SIGMOD Int'l Conference on Management of Data, 1997, pp. 255-264.

[7] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth: "From Data Mining to Knowledge Discovery: An Overview". In Advances in Knowledge Discovery and Data Mining, Fayyad U, et. al. eds, AAAI Press, 1996, pp. 1-35.

[8] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus C. J: "Knowledge Discovery in Database: An Overview", In GTE Laboratories paper, 1996, pp. 1-27.

[9] J. Han, J. Pei, and Y. Yin: "Mining Frequent Patterns without Candidate Generation". In Proc. of the ACM SIGMOD Int'l Conference on Management of Data, 2000, pp. 1-12.

[10] J. Liu, Y. Pan, K. Wang, and J. Han: "Mining Frequent Itemsets by Opportunistic Projection". In Proc. of the ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining, 2002, pp. 229-238.

[11] S. Orlando, C. Lucchese, P. Palmerini, R. Perego, and F. Silvestri: "kDCI: a Multi-Strategy Algorithm for Mining Frequent Sets". In Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, 2003.

[12] R. Osmar. Zaïane and Mohammed El-Hajj: "COFI-tree Mining: A New Approach to Pattern Growth with Reduced Candidacy Generation". In Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, 2003.

[13] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang: "H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases". In Proc. of the IEEE ICDM Int'l Conference on Data Mining, 2001, pp. 441-448.

[14] B, Raacz: "Nonordfp: An FP-growth Variation without Rebuilding the FP-tree". In Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, 2004.

[15] A. Savasere, E. Omiecinski, and S. Navathe: "An Efficient Algorithm for Mining Association Rules in Large Databases". In Proc. of the Int'l Conference on Very Large Databases (VLDB), 1995, pp. 432-444.

[16] Y. G. Sucahyo and R. P. Gopalan: "CT-PRO: A Bottom-Up Non Recursive Frequent Itemset Mining Algorithm Using Compressed FP-Tree Data Structure". In Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, 2004.

[17] T. Uno, T. Asai, Y. Uchida, H. Arimura. LCM: "An Efficient Algorithm for Enumerating Frequent Closed Item Sets". In Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, 2003.

[18] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li: "New Algorithms for Fast Discovery of Association Rules". In Proc. of the IEEE KDD Int'l Conference on Knowledge Discovery and Data Mining, 1997, pp. 283-286.

Evolution Shelf: Exploiting Evolution Styles within Software Architectures

Olivier Le Goaer, Mourad-Chabane Oussalah, Dalila Tamzalit
University of Nantes
LINA Laboratory
2 rue de la Houssiniere, F-44000 Nantes, France
olivier.le-goaer@univ-nantes.fr

Abdelhak-Djamel Seriai
Ecole des Mines de Douai
Departement I.A
941 rue Charles Bourseul, F-59500 Douai, France
seriai@ensm-douai.fr

Abstract

Despite that reuse libraries are now well adopted during software development step, software evolution step is not yet covered by this kind of beneficial approach. In this paper we present the "evolution shelf", a generic infrastructure to achieve for-reuse and by-reuse techniques within the field of software evolution. The basic idea behind that is to propose and encourage the reuse of recurring and reliable expertises to achieve the structural evolution of a software system at the architectural level. For that purpose, the shelf assists architects in classifying, storing and selecting reusable architectural evolutions. The underlying concept that we propose to capitalize the expertises is called "evolution style", mixing a syntactic and a semantic description format. These ideas form a core for a long-term vision in which it is possible to build a business model of evolution-off-the-shelf (EOTS) with the special objective to decrease the efforts and the risks related to the evolution activities.

1. Introduction

Traditionally, the primary focus of reuse research has been on the reuse of software entities, such as objects or components [11], often at code-level but also at more abstract levels. While there have been significant improvements in reuse technology and methods, these artifacts are not the only ones that can be profitably reused. In this paper we describe an approach and supporting infrastructure for a class of skill reuse – namely, architectural evolution reuse.

An increasingly important requirement for software-based systems is the ability to embrace change. From a theoretical point of view, embracing change aims at integrating the change as a natural ingredient of modern software systems and be prepared to challenge it. From a practical point of view, embracing change aims at reacting to unanticipated events by the way of predictable means. Especially, in this research, we deal with the evolution of the structure of component-based architectures. An architectural design is concerned with the gross decomposition of a system into a set of interacting components [15, 10]. At this level of abstraction, key issues include the assignment of functionality to design elements, protocols of interaction, system extensibility, and broad system properties such as throughput, schedulability, and overall performance. The reuse problem for architectural designs then becomes how to exploit the basic elements of architectural design (large-scale components and connectors), as well as common processes for their evolution over time.

But what exactly is evolution reuse and how can it be exploited ? The basic idea is to capitalize know-how on a specific domain, providing specific facilities for a fairly narrow class of system. For this purpose, we lean our work on the concept of architectural style [18, 14], used for representing and reusing architectural designs and design fragments for a family of architectures. An architectural style is a powerful design artifact which captures the aspects of a particular domain of application and can provide assurance that elements built following the stylistic guidelines are interoperable. A crucial observation here is that architectural styles are analogous to domain-specific languages [1]. Throughout refinements of a previous work [16], we propose the concept of evolution style for the enrichment of the concept of architectural style, and to turn it into a run-time ar-

tifact. The evolution styles are used for representing and reusing the evolution of architectural designs. Further, the use of evolution style supplements traditional mechanisms for classifying evolution processes, storing those processes in a repository, and narrowing the search space to more accurately locate potential process matches in a given context.

We begin by contrasting our approach to existing work in the area of reuse and knowledge engineering, in the field of software architecture. In Section 3, we show how the concept of evolution style can extend the concept of architectural style. As a demonstration of how these ideas can be used, in Section 4 we describe a prototype infrastructure called the "Evolution Shelf" which provides a repository for reusable architectural evolutions. We give some concluding remarks in Section 5.

2. Related Work

The research presented here lies at the intersection of two related areas: the reuse engineering and the knowledge engineering. In our context, we put a special focus on the software architecture field.

Some of the more impressive examples of reuse today involve a strong component of design reuse that can be separated into three classes of granularity of abstraction [4]: architectural patterns and styles, design patterns and idioms. Prominent examples include the book of object-oriented design patterns by Gamma et al. [8] or the book of architectural patterns by Bushman et al. [5]. Broadly speaking, our work is motivated by the same concerns – providing high-level abstractions for evolving systems based on previous efforts and proven know-how. While we also exploit architectural commonalities (in our use of "architectural style"), we address an orthogonal issue. In our context, the term of "style" refers to the more formal aspect of the descriptions of solutions and thus the possible construction of an automated support. This last point naturally introduces a second crucial aspect: the desired organization of such an amount of knowledge into libraries, able to support efficient reuse techniques. As an example, a preliminary classification of architectural styles was presented in Shaw and Clements [17], in which a two-dimensional, tabular classification strategy was used with control and data issues as the primary axes, organized by the following categories of features: which kinds of components and connectors are used in the style; how control is shared, allocated, and transferred among the components; how data is communicated through the system; how data and control interact; and, what type of reasoning is compatible with the style. The primary purpose of the taxonomy was to identify style characteristics, rather than to assist in their comparison. Hence, organizing the classification in that fashion does not help a designer find a style that corresponds to their needs. As another example, Zimmer [20] organized design patterns using a graph based on their relationships, making it easier to understand the overall structure of the patterns in the Gamma et al. catalog. The classification was based exclusively on derivation or use relationship, but no infrastructure was provided to exploit it. In [13], Monroe and Garlan built a software repository that assists designers in selecting design elements and patterns based on stylistic information and design constraints. Their repository was evaluated on the Aesop system with a great size of elements and can be queried with a specific language based on stylistic attributes, as well as traditional string and keyword matching. Like reported in [7], despite that there has been disagreement in the reuse research community about the importance of libraries for reuse, the reuse activity shows that in order to be reused an asset must be available, findable, and understandable. A reuse library supports all of these and should be used in the field of evolution engineering.

3. Style-based Evolution Model

Basically, architectural evolution includes modifications for general purposes, in terms of additions, removals and modifications on first-class entities, namely, the component, the connector, the interface (derived into port and role) and the configuration. Domain-specific architectural evolution extends the aforesaid principle to take into account the specific aspects provided by an architectural style. For example, the *Pipe&Filter* architectural style supports particular evolutions such as *Become-a-sink*, for setting a filter solely as a data consumer by deleting all its output interfaces. Hence, the evolution styles we deal with in this paper are specifically designed for a family of applications that satisfy a given architectural style. In this section we explain the description format of an evolution style and we provide a small example.

3.1. Specification of Evolution Styles

The language we propose is captured by the metamodel given in Figure 1, represented in an object-oriented context with the UML notation. Within this metamodel, an architectural style encompasses:

- a set of types of architectural element, derived into types of configuration, types of component, types of connector and types of interface. An architectural style is simply viewed as a type of configuration. These types provide a domain-specific design vocabulary.

- constraints that define how components can be integrated. Constraints may be topological, behavioral, communication-oriented. In this research we are only

interested by rules that govern the structure of the architecture, that is, the topological rules.

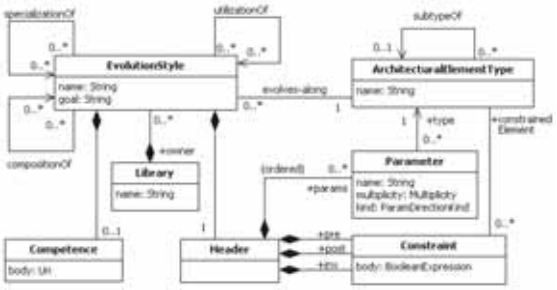


Figure 1. Metamodel of the concept of evolution style (UML class diagram).

Any of these types of architectural element is likely to evolve along some predefined archetypes, the so-called evolution styles. The two basic information of an evolution style are an evocative name and the textual description of its goal. Then, the proposed specification of an evolution style is formed by two complementary parts: the header and the competence. From a theoretic point of view, in the spirit of the "Components of Expertise" approach [19], the header denotes an evolution task while the competence denotes a problem solving method (PSM) to achieve the task. From a practical point of view, the purpose of the header is to publish the interface and the behavior of the evolution while the purpose of the competence is to give an implementation to realize the header following a particular strategy. This decoupling follows the information hiding principle and enables more flexible specifications.

Syntactic description – The header publishes the interface of the evolution by mentioning an ordered set of parameters typed with the types of design elements provided by a particular architectural style. Each parameter has a multiplicity (i.e., multi-valued parameters are supported) and a direction kind (In/Out/InOut/Return). The header also publishes the behavior of the evolution by mentioning three assertions: a precondition, an invariant and a postcondition. These assertions are constraints expressed with a first order predicate logic similar to UML's OCL, augmented with a small set of architectural functions. The precondition and the postcondition are user-defined while the invariant is virtually imported from the one imposed on the considered design element. Constraints are essential both to preserve the initial design choices and to leave the architecture in a consistent state after its evolution.

The competence realizes the header by a particular implementation. Unlike the header, sometimes the competence cannot be identified. When the competence part is missing, the evolution style is abstract (and plays a major role in knowledge structuring). Otherwise, the evolution style is concrete and its competence part refers to an operation tied to a technological context: a C function or procedure, a JAVA object method or a model transformation, etc. Hence, to have a technologically-neutral competence, the concrete implementation is considered as an external resource located with an URI (Uniform Resource Identifier). Of course, an implementation is required to turn the evolution style into an executable form.

Semantic description – The evolution styles can be interrelated in three various ways: with the (i) specialization relation, with the (ii) composition relation and with the (iii) utilization relation. These relations bring a semantic dimension to the purely syntactic description of the concept of evolution style. The specialization and the composition define hierarchies amongst a set of evolution styles: the former defines a conceptual hierarchy (i.e., general vs. specialized evolutions) while the latter defines a descriptive hierarchy (i.e., coarse-grained vs. fine-grained evolutions). The utilization does not define any hierarchy amongst a set of evolution styles but enables important collaborations. From an operational viewpoint, these relations play different roles, but all convey invocations – or calls. The specialization is useful at design-time to build new description from older ones. A specific inheritance mechanism allows the user to overload a header and/or to override a competence. The composition is useful at design-time to assembly descriptions into a composite one and the utilization is crucial at run-time to properly propagate the impacts of an evolution throughout a chaining of styles. Finally, a set of inter-related evolution styles designed for a particular architectural style forms an evolution library. From the end-user point of view, such a library is a black-box that is intended to be enriched and queried via an infrastructure specifically designed to hold it, so-called a "shelf".

3.2. Sample Illustration

We illustrate a client-server architectural style on which a preliminary evolution library is designed, inspired from a scenario found in [6]. The architectural style is described using the ACMEADL [9] and the ARMANI constraint language [12]. On the other hand, the evolution styles are described using the UML 2.0 language extended with a specific profile (i.e., stereotypes, tagged values and constraints) that we have developed for representing the content of the evolution libraries in a diagrammatic fashion.

The first step specifies a generic *client-server* architectural style (called a family in ACME). It defines a set of component types: a client type (`ClientT`), a server group

type (ServerGroupT) and a server type (ServerT). It also defines a connector type (LinkT). Constraints on the style (appearing in the definition of LinkT) guarantee that the link has only one role for the server and more than one role for the client. Other constraints, not shown, define further structural rules (for example, each client must be connected to a server). The corresponding code snipet is given in Figure 2.

```
1: Family ClientServer = {
2:     Component Type ClientT = {...};
3:     Component Type ServerT = {...};
4:     Component Type ServerGroupT = {
5:      Property AverageLoad : float
6:       ...
7:     };
8:     Role Type ClientRoleT = {...};
9:     Role Type ServerRoleT = {...};
10:    Connector Type LinkT = {
11:      invariant size(select r : role in Self.Roles |
                declaresType(r, ServerRoleT)) == 1;
12:      invariant size(select r : role in Self.Roles |
                declaresType(r, ClientRoleT)) >= 1;
13:      Role ClientRole1 : ClientRoleT;
14:      Role ServerRole : ServerRoleT;
15:    };
16: }
```

Figure 2. A Client-Server architectural style in ACME.

The second step specifies a *MoveClient* evolution style designed for the client-server architectural style above. The purpose of this evolution is to reconnect a client to another server group. Practically, the latter is applied to a client and deletes the role currently connecting the client to the connector that connects it to a server group. This style also performs the necessary attachment to a connector that will connect it to the server group passed in as a parameter. The diagram on Figure 3 shows the *MoveClient* evolution style but also some of its adjoining evolution styles, through different semantic links. Due to place restrictions, the constraints (usually expressed in OCL) and the goals (usually expressed with a UML note) are not shown on the diagram.

At glance, moving a client to another group of servers is viewed as an evolution decomposable into disconnecting a client from an older group and then reconnecting it to a new group. Accordingly, disconnection or connection propagates the evolution to update the load of the group of servers involved. All these evolution styles are specializations of an abstract evolution style that is the "top" of the specialization hierarchy. From now, the *MoveClient* evolution style can be specialized, composed or utilized in order to define more complex evolution strategies. However, it remains to provide reuse facilities to experts for enabling them to create further expertises more easily.

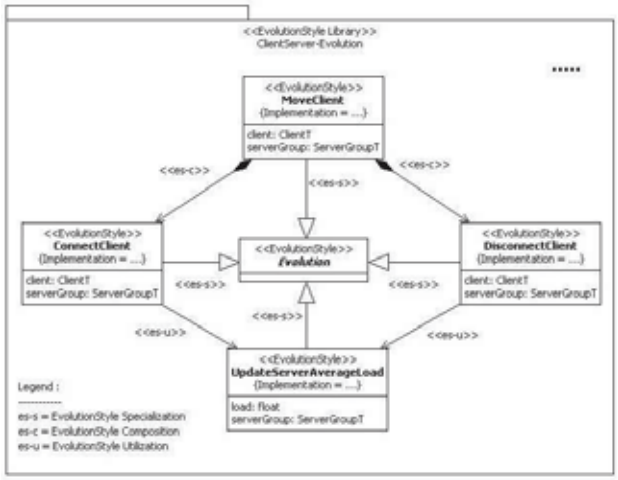


Figure 3. UML-based evolution style diagram depicting an excerpt of the evolution library associated to the client-server architectural style.

4. The Evolution Shelf

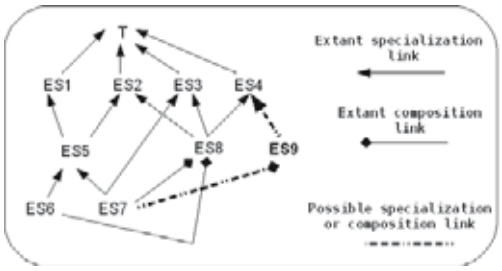
The Evolution Shelf is an infrastructure to support the classification, storage and retrieval of evolution styles. We now describe the strategy taken to implement this functionality. Many of the underlying mechanisms that we use are not new. Our intention is to leverage well-known repository techniques while updating them to support and exploit the concept of evolution style.

4.1. Classification

The evolution styles grouped into libraries and stored on the shelf are organized along three semantic relations introduced in Section 3.1. The primary benefits that the use of evolution styles brings to this taxonomical scheme is that users are able to determine a lot about a reusable style based solely on its three-dimensional organization. From a knowledge-oriented standpoint, a library is a knowledge base which organizes knowledge fragments to deal with some recurring problem/solution pairs in the software evolution field. From an operational standpoint, a library is a directed graph where the nodes are evolution styles and the edges are instances of the three semantic relations. Views can be created on the graph in order to put the focus on subgraphs from a given semantic perspective, namely the specialization graph, the composition graph and the utilization graph.

The term "classification" is not used to refer to a hierarchical structure; it refers to the mechanism that aims at

Figure 4. Result of the classification of an evolution style ES9 into an evolution library.



determining the relevant position of an evolution style in a hierarchical structure. Precisely, the classification process computes the position of a new style into the specialization graph and the composition graph (See Fig.4). As stated before, the utilization graph does not define hierarchy and hence is not concerned by this process. The process relies on a subsumption function – widely used in the area of concept lattices – to discover a semantic relation between two non-connected evolution styles. The storage and retrieval mechanisms are both based on the same classification process. Let us now detail the approach.

4.2. Storage

Due to the great size of a library, the addition of a new evolution style is difficult: to find its correct position is complicated because the user does not know exactly the whole base entities, and might thus introduce some inconsistencies. It is proper to have an automation of the structural acquisition process: it will be used to find a correct position both in the specialization hierarchy and in the composition hierarchy. Notice that a library can be entirely built from scratch following this process.

The general storage process to insert a new evolution style X into a hierarchy operates according to a three-phase loop:

1. retrieve the most specific subsuming evolution styles that subsume X (the "immediate ascendants" of X, denoted $MSS(X)$ further),
2. retrieve the most general evolution styles subsumed by X (the "immediate descendants" of X, denoted $MGS(X)$ further),
3. (possibly) update the links in the hierarchy while inserting the new evolution style; the link updates may generate loop in this process.

In the first step, the set of evolution styles that subsume X is computed thanks to a top-down traversal of the hierarchy. In the second step, the set of evolution styles that are subsumed by X are determined by, roughly, exploring the descendants of the $MSS(X)$ computed in the first step. The specialization graph defines a single hierarchy rooted at a "Top" node (denoted $\top$), from which the step 1 is started once. The composition graph may define multiple hierarchies where each node is a potential root, from which the step 1 is repeated. From an algorithmic standpoint, the composition-centered classification procedure is nested in the specialization-centered classification procedure.

Broadly speaking, having an infrastructure for the maintenance of various semantic links between knowledge base entities is very important. Indeed, these links hold up the consistency and the correctness of library, and are challenged whenever an insertion of a new evolution style occurs. A management only performed by the user could be very dangerous, even impossible, whenever the size of the library reaches a certain threshold.

4.3. Retrieval

Besides to a traditional keyword-based search technique, a query is represented as an abstract evolution style Q. We remember that an abstract evolution style does not define any competence. This is not surprising since the competence is the important information expected by the user. One should notice that this uniformity has at least three benefits: (i) the user does not need to learn a further language for querying the libraries, (ii) the infrastructure is "code-saving" and (iii) tools built on top of the concept of evolution style (for exportation, visualization, etc.) can be reapplied to the queries. From our concern, this approach closely related to description logics used in the database domain [2, 3], is used for query purpose with the special objective to produce more than "Yes or No" results. For that reason, the query evaluation consists of "situating" Q both into the specialization graph and the composition graph. In both cases, the result of the query is obtained thanks to the computing of $MSS(Q)$ and $MGS(Q)$, excluding the abstract styles (i.e., without competence and hence not executable). Let us informally summarize the various situations:

- Case 1: The results exactly satisfy the request. This case arises when $MSS(Q) = MGS(Q)$, both in the specialization *and* the composition hierarchy. In other words, a single evolution style satisfies the query and hence it can be reused with confidence.
- Case 2: The results satisfy the request in half. This case arises when $MSS(Q) = MGS(Q)$, in the specialization *or* the composition hierarchy. In other words, a single evolution style satisfies the query from one given semantic perspective and hence it can probably be reused.

- Case 3: The $MSS(Q)$ computed in the specialization hierarchy are more general evolution styles. Any one of them can be reused without errors because they can substitute to the query (due to "style substyping").
- Case 4: The $MSS(Q)$ computed in the composition hierarchy are more fine-grained evolution styles. Hence, they are intended to be reused "together".
- Case 5: No results satisfy the request. This case arises when $MSS(Q)$ and $MGS(Q)$ are empty sets, both in the specialization *and* the composition hierarchy. This means that the user is facing a problem not covered by the library.

At glance, cases 1 and 5 provide a binary result (i.e., Success or Failure), like provided by any repository search techniques. In contrast, cases 2,3 and 4 provide an intermediate range of probable results that extends the reuse possibilities one step further.

5. Concluding Remarks

Evolution reuse would appear to be one of the more promising avenues for improving the prospects for software reuse. We have shown in this paper that our motivation is to provide abstractions for encapsulating evolution expertise independently of any technological environment. To a larger extent, the basic idea behind that is to use the concept of evolution style as an neutral interchange format to capitalize and transfer knowledge about evolution tasks.

Architectural styles leverage the construction phase of software architectures. We have been exploring a different but complementary approach: Evolution styles leverage the evolution phase of software architectures. The primary contribution of this work is the integration of the concept of evolution style with traditional mechanisms for performing standard software reuse tasks. Finally, to integrate our approach into realistic developments, the Evolution Shelf we have presented in this paper should be embedded into conventional CASE tools.

References

[1] Domain-specific software architecture (dssa) frequently asked questions (faq). *SIGSOFT Softw. Eng. Notes*, 19(2):52–56, 1994.

[2] A. Borgida. Description logics in data management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–682, 1995.

[3] N. Boudjlida. Knowledge in interoperable and evolutionary systems. In *KRDB*, 1995.

[4] F. Buschmann and R. Meunier. A system of patterns. pages 325–343, 1995.

[5] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-oriented software architecture: a system of patterns*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[6] S.-W. Cheng, D. Garlan, B. R. Schmerl, a. P. S. Jo B. Spitnagel, and P. Steenkiste. Using architectural style as a basis for system self-repair. In *WICSA 3: Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture*, pages 45–59, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V.

[7] W. B. Frakes and K. Kang. Software reuse research: Status and future. *IEEE Trans. Softw. Eng.*, 31(7):529–536, 2005.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[9] D. Garlan, R. T. Monroe, and D. Wile. Acme: Architectural description of component-based systems. In G. T. Leavens and M. Sitaraman, editors, *Foundations of Component-Based Systems*, pages 47–68. Cambridge University Press, 2000.

[10] D. Garlan and M. Shaw. An introduction to software architecture. Technical report, Pittsburgh, PA, USA, 1994.

[11] B. Meyer. *Reusable software: the Base object-oriented component libraries*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[12] R. Monroe. Capturing software architecture design expertise with armani. Technical report, School of Computer Science, Carnegie Mellon University, 2001.

[13] R. T. Monroe and D. Garlan. Style-based reuse for software architectures. In *ICSR '96: Proceedings of the 4th International Conference on Software Reuse*, page 84, Washington, DC, USA, 1996. IEEE Computer Society.

[14] R. T. Monroe, A. Kompanek, R. Melton, and D. Garlan. Architectural styles, design patterns, and objects. *IEEE Softw.*, 14(1):43–52, 1997.

[15] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52, 1992.

[16] A. Seriai, M. C. Oussalah, D. Tamzalit, and O. L. Goaer. A reuse-driven approach to update component-based software architectures. In *IRI*, pages 313–318, 2006.

[17] M. Shaw and P. C. Clements. A field guide to boxology: Preliminary classification of architectural styles for software systems. In *COMPSAC '97: Proceedings of the 21st International Computer Software and Applications Conference*, pages 6–13, Washington, DC, USA, 1997. IEEE Computer Society.

[18] M. Shaw and D. Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[19] L. Steels. Components of expertise. *AI Mag.*, 11(2):30–49, 1990.

[20] W. Zimmer. Relationships between design patterns. pages 345–364, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

Coverage-Based Testing Using Qualitative Reasoning Models

Harald Brandl Gordon Fraser Franz Wotawa*

Institute for Software Technology
Graz University of Technology
8010 Graz, Austria
{brandl,fraser,wotawa}@ist.tugraz.at

Abstract

The use of explicit environment models for test case generation allows to exercise a system under test in ways that are not possible when only considering the system by itself. Modeling a system's environment, however, is not an easy task. Qualitative Reasoning (QR), which is a well known artificial intelligence technique to represent and reason about physical behavior, is well suited to describe such environments. In this paper, we define a set of coverage criteria that allow an evaluation of how well the behavior of a system's environment and interactions with the environment are exercised with a given test suite. Using a popular QR modeling tool, we show how to derive test cases satisfying these coverage criteria automatically, and illustrate these results on an example application.

1 Introduction

Software testing remains the most important technique to ensure sufficient quality of software. The amount of software included in everyday products constantly increases, and so does its complexity. Especially when considering safety relevant applications like many of the control units in modern cars, the need for thorough testing methods becomes apparent. Unfortunately, testing is a very complex task, particularly in the context of reactive and embedded systems. Here, the complexity of testing is further increased because there is a lot of interaction between a system and its environment. A high level of interaction between system and environment requires extensive testing of the system's reactions to environment stimuli. However, many available testing techniques consider only the system under test, and not its environment.

To overcome this drawback, we propose a methodology for testing embedded systems in terms of their environmental interactions. Qualitative Reasoning (QR), a well-known artificial intelligence technique, is used to describe the environmental behavior in an abstract form. Qualitative abstraction maps large or infinite domains to discrete intervals, which enables QR engines to infer all possible behaviors from a model. The possible behaviors are represented as a transition system – a classical modeling formalism often used for automated testing. The benefit of QR is the ability to reason about domain intervals and physical behavior.

Given a transition system derived from a QR model, we are facing the classical testing problem of which test cases to select out of a very large possible number. In this paper, this problem is solved by defining a set of coverage criteria with information provided by the QR model. This allows to determine how well a system is tested with respect to its environment. As existing testing techniques possibly do not perform well on testing the environmental interactions, we describe a method to automatically derive test suites for the presented coverage criteria. The resulting technique can be seen as a complementary technique for traditional testing methods.

In detail, the contributions of this paper are as follows:

- Testing based on environmental models exercises a system under test in ways that might not be possible with traditional behavioral models.
- QR, a well-known AI technique to represent and reason about physical systems, is adapted to software testing.
- A set of coverage criteria for QR models allows evaluation of test suites with regard to the environment.
- A method for automated coverage based test case generation for QR models is proposed.

This paper is organized as follows: First we give a brief introduction to QR modeling in Section 2, then we define several coverage criteria based on QR models in Section 3. In Section 4 we describe how to automatically derive test

*Authors are listed in alphabetical order.

cases for the discussed coverage criteria and present the results of applying the test case generation to a case study model. Finally, we consider related research in Section 5 and conclude the paper in Section 6.

2 Qualitative Reasoning

The initial idea that lead to the development of QR was to formalize and finally implement commonsense reasoning about physical systems. The objective was to enable a computer to use the same reasoning capabilities as humans have. Typical application areas include trouble shooting of physical systems and tutoring systems. Because QR offers techniques for representing physical knowledge in an abstract way it fits well for testing embedded systems in terms of their environmental interactions. In the case of testing embedded systems we consider the system software as a black box with an interface to the environment comprising sensors and actuators; this interface has to be tested with respect to all possible interactions.

In most cases system environments have infinite rather than finite domains like a switch with four positions. Although infinite domain environments are made finite by quantization during Analog to Digital conversion the state space is still too large for enumeration. In addition, such environmental quantities with infinite domains show continuous behavior that can be expressed by differential equations. We use Qualitative Reasoning (QR) to express environmental behavior in an abstract form. Qualitative abstraction maps large or infinite domains to intervals separated by points (landmarks), which enables QR engines to infer all possible behaviors from a model. This is in contrast to traditional testing techniques, where the test engineer has to develop test scenarios that reflect possible traces of the environment and thus very likely forgets some important traces.

Modeling in QR might follow different branches like qualitative simulation [8] or qualitative process theory [6]. In this paper we use qualitative process theory as the underlying QR modeling paradigm because of the general availability of tools like Garp3 [4]. For a detailed description of the functionality and QR modeling capabilities of Garp3 we refer the reader to [2, 5]. In order to be self contained we briefly introduce the basic concepts of modeling in QR and Garp3. Garp3 models comprise a set of model fragments. Two main types of fragments can be distinguished: static and process model fragments. Static fragments represent behavior that is invariant with regard to time, such as proportional relations between quantities, like, for example *"the amount of water in a vessel is proportional to the water level"*. A dynamic fragment introduces changes via influences between quantities, for example *"a positive flow rate into a vessel will increase the amount of liquid and hence the liquid level over time"*.

The behavior of a system comprises at least one but usually more model fragments that are activated when certain boundary conditions are met. Garp3 adds the consequences of a model fragment as new facts to the knowledge base, unless they contradict existing facts. Model fragments enable the designer to partition the system domain into qualitative equivalence classes that capture certain behavior. During simulation the set of fragments collected in a library changes between being active and inactive as the system evolves over time.

Within a model fragment the main modeling primitives are entities, quantities, proportionalities, and a set of ordinal relations. In dynamic model fragments there are additional influences. Entities are the components of the system that have certain properties expressed through associated quantities. For example, an entity "tank" has quantities like "level" and "inflow rate". Proportionalities establish a mathematical relation between two quantities in the form of a monotonic increasing or decreasing function. The notation P+ (Q1,Q2) expresses that a change of Q2 causes a change of Q1 in the same direction. A proportionality with a minus sign states that a change of the cause quantity induces a change in the opposite direction of the effected quantity.

Ordinal relations called inequalities provide means to constrain possible behavior. Influences cause dynamic changes of the system and provide means for integration. For instance, I+ (Q1,Q2) means that the value of Q2 determines the change of direction of Q1. If Q1 is positive Q2 increases, if Q1 is zero Q2 does not change, and if Q2 is negative Q1 decreases. The graphical notation used in Garp3 states relations with arrows between quantities.

The initial state of the system is captured with scenarios. This initial state and the model fragments serve as inputs to the simulation engine. Simulation is used to generate the behavior of a QR system. The simulation engine derives everything that does not contradict the boundary conditions, i.e, inequalities between quantities. QR models can only describe systems with continuously changing quantities, as stated by the continuity law [6]. In general, model creation is an iterative process: One has to find the right level of abstraction and check if the simulation output satisfies the requirements. If there are discrepancies, the model has to be adapted and simulated again.

Example: Consider the two-tank system from Figure 1. The two tanks are connected via a pipe at the bottom. Water can flow in both directions through the pipe. The flow rate depends on the difference of water levels. The control system has to hold the water level of *tank2* constant at a specified height while the water level and inflow rate of *tank1* varies. The control system can set the inflow to *tank1* and the outflow of *tank2* via controlling the valves. The water levels of both tanks are inputs of the control system. We use

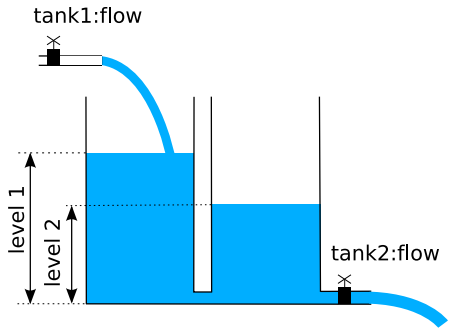


Figure 1. Two-Tank Example System.

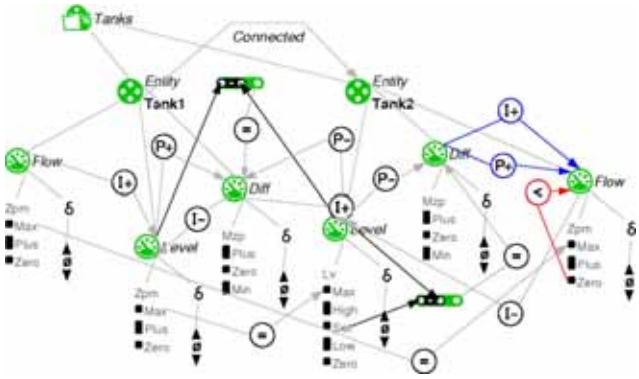


Figure 2. Model Fragment of the Two-Tank System.

this model as a case study in this paper. Figure 2 shows the Garp3 model fragment representing the physical relations and the control loop of the two-tank system.

The water level of *tank1* is the integral of the inflow rate over time, represented via a positive influence from *flow* to *level*. *Tank1* has an auxiliary quantity *diff* for calculating the difference in water levels. This difference determines the water flow through the pipe while influencing the water levels of both tanks. The quantity space of *level* in *tank2* contains a *set* point for controlling the water level. We use the auxiliary quantity *diff* of *tank2* to calculate the control deviation. The control loop is closed via a P and an I arrow to the actuator controlling the outflow valve resulting in a PI-controller. According to control theory the proportional part ensures quick response to changes and the integration part eliminates permanent control deviations. The setting of the outflow valve has a negative influence to the water level of *tank2*. Finally, equalities between the maximum tank water levels and valve flow rates makes them comparable while reducing ambiguity.

3 Coverage Criteria for QR Models

In general, one of the main problems one has to face when generating test cases is which test cases out of a large or infinite set to choose. Coverage criteria have been successfully used as stopping criteria when generating test cases, to evaluate test suites, and to automatically derive test cases.

A coverage criterion describes a set of items that the testing process should exercise. Many different coverage criteria have been defined based on both source code and specifications. For example, given a transition system we might aim to cover each transition of every state at least once.

When using explicit environment models the question of which test cases to select remains. Therefore, we define a set of coverage criteria for QR models: domain coverage, delta coverage, full delta coverage, as well as the traditional state and transition coverage.

The simulation of a QR models results in a state space representation of all possible behaviors that may evolve over time, starting with an initial scenario. This output is a QR transition system (QR TS) $M = (S, T, s_0, Q, qs, QS, v, \delta)$, where S is the set of states, T is the transition relation $T \subseteq S \times S$, and $s_0 \in S$ is the initial state.

Every quantity $q \in Q$ has an associated quantity space in the domain of quantity spaces QS, and the function $qs : Q \rightarrow QS$ maps each quantity to its associated quantity space. Each state $s \in S$ binds each quantity to a distinct value and delta. The value v for quantities in Q and states in S is defined as $v : S \times Q \rightarrow qs(Q)$, and the delta δ is defined as $\delta : S \times Q \rightarrow \{min, zero, plus\}$. The $delta$ of a value stands for its direction of change over time, $\delta \sim \frac{\partial value}{\partial t}$.

Below we define a set of different coverage criteria. A coverage value according to these coverage criteria can be measured as the ratio of covered items to items in total as defined in the coverage criterion. As will be described in Section 4, a test case created from a QR TS is itself a QR TS $t = (S_t, T_t, s_{0,t}, Q_t, qs_t, QS_t, v_t, \delta_t)$. A test suite $\mathcal{T}$ is a set of test cases.

In a QR transition system, the state of the environment is given by a value assignment to the model's quantities. Each quantity $q \in Q$ has a finite domain, its quantity space: $qs(q) = \{d_0, d_1, \ldots, d_n\}$, therefore it is feasible to require each quantity to take on all of its values. This coverage criterion is called *Domain Coverage*.

Definition 1 (Domain Coverage) *A test suite $\mathcal{T}$ achieves domain coverage, if for each quantity $q \in Q$, q there is a test case $t = (S_t, T_t, s_{0,t}, Q_t, qs_t, QS_t, v_t, \delta_t) \in \mathcal{T}$ for each $d \in qs(q)$, such that $s \in S_t : v(s, q) = d$.*

Because testing all possible environment states might be impractical, domain coverage offers a compromise by en-

suring that each possible domain value occurs at some point. That is, given a test suite that satisfies domain coverage we know that each possible environment value has been exercised at least once.

As quantities change their values according to the description given in the QR Model, errors might only be detected when considering value changes. In QR models, the direction of change of a quantity is given by its delta, which can be min, $zero$, or $plus$. Consequently, we define *Delta Coverage* such that each quantity has changed its value in every direction at least once.

Definition 2 (Delta Coverage) *A test suite $\mathcal{T}$ achieves delta coverage, if for each quantity $q \in Q$, q there are test cases $t_1 \in \mathcal{T}$, $t_2 \in \mathcal{T}$, and $t_3 \in \mathcal{T}$ such that $s_1 \in S_{t_1} \wedge \delta(s_1, q) = min$, $s_2 \in S_{t_2} \wedge \delta(s_2, q) = zero$, and $s_3 \in S_{t_3} \wedge \delta(s_3, q) = max$.*

Delta coverage ensures that every relevant quantity eventually changes its direction. Note that in Definition 2 t_1, t_2, and t_3 can be the same test case. Changes of direction adhere to the continuity law [6]. This means that a quantity has to increase, stay steady, and then decrease or vice versa. Sudden, non-continuous jumps on domain values or deltas cannot happen.

Delta coverage can miss cases where some value change causes an error but a different value change for the same delta is chosen for testing. Consequently, we define *Complete Delta Coverage* as a stricter variant of delta coverage.

Definition 3 (Complete Delta Coverage) *A test suite $\mathcal{T}$ achieves complete delta coverage, if for each quantity $q \in Q$ with quantity space $qs(Q)$, there are test cases t_1, t_2, and t_3 for each $d \in qs(Q)$ such that $s_1 \in S_{t_1} \wedge \delta(s_1, q) = min \wedge v(s_1, q) = d$, $s_2 \in S_{t_2} \wedge \delta(s_2, q) = zero \wedge v(s_2, q) = d$, and $s_3 \in S_{t_3} \wedge \delta(s_3, q) = max \wedge v(s_3, q) = d$.*

Complete delta coverage is a combination of delta and domain coverage demanding that every domain value of a quantity changes its direction. Depending on the number of quantities and the sizes of their quantity spaces, complete delta coverage might be hard to achieve. Complete delta coverage might also contain infeasible test goals; for example it might not be possible to decrease a value (delta min) at the lower bound of a quantity's domain.

In addition to the above coverage criteria, we can apply traditional coverage criteria for transition systems to the QR TS. *State coverage* requires that each state of the QR TS is visited at least once, and *Transition coverage* requires that every transition of the QR TS is executed.

Definition 4 (State Coverage) *A test suite $\mathcal{T}$ achieves state coverage for QR TS $M = (S, T, s_0, Q, qs, QS, v, \delta)$, if for every $s \in S$ there is a test case $t = (S_t, T_t, s_{0,t}, Q_t, qs_t, QS_t, v_t, \delta_t) \in \mathcal{T}$ such that $s \in S_t$.*

Definition 5 (Transition Coverage) *A test suite $\mathcal{T}$ achieves transition coverage for QR TS $M = (S, T, s_0, Q, qs, QS, v, \delta)$, if for every $(s, s') \in T$ there is a test case $t = (S_t, T_t, s_{0,t}, Q_t, qs_t, QS_t, v_t, \delta_t) \in \mathcal{T}$ such that $(s, s') \in T_t$.*

4 Automated Coverage Based Test Case Generation

So far we have defined coverage criteria to evaluate existing test suites. In this section we describe how to automatically derive test suites that satisfy these coverage criteria using QR models. For this we use a recently proposed method [3] for deriving test cases from QR models with formally defined test purposes.

4.1 Test Case Generation for QR Models

In general, test case generation with test purposes requires a test engineer to write test purposes in a formal notation. In the case of QR models, the approach described in [3] uses properties defined with regard to quantities to label the transitions of a QR TS. These properties are used to turn the QR TS into a labeled transition system, and to describe test purposes. For instance a property $a := tank1 : level\ gt\ zero$, referenced by symbol a, denotes that the water level of tank1 is greater than zero. The test purpose is specified via a regular expression over property symbols. For example $p\{3\}q$ denotes a sequence of states, where property p holds in the first three states followed by a state which satisfies property q. The regular expression is converted to a finite automaton, and the synchronous product of QR TS and test purpose results in a Complete Test Graph (CTG), from which test cases are extracted.

As a QR TS can be nondeterministic, special care has to be taken to ensure controllability, i.e., to make sure that a test case can handle all decisions between different inputs for the implementation. Consequently, test cases are not linear sequences but transition systems that can handle alternative outputs. Given a CTG, test cases are created by searching backwards to the start state and forward to an accept state. Then the path is traversed and all parts of the CTG that are reachable considering the implementation's nondeterministic outputs are added to the test case.

A resulting test case is a QR TS, which includes all information needed for its execution in its states. Each QR state comprises a set of quantities with their current values and deltas. For test case execution, these abstract values have to be mapped to concrete quantity values.

During test case execution, the concrete values are updated regularly based on a time step Δt according to the current state. The behavior of the output quantities decides

which transitions are taken, and a state is left as soon as it becomes inconsistent with the input quantities it controls and the observed output quantities. If there is no matching successor state that is consistent with the observed outputs, then the implementation fails the test case. When an accepting state is reached the implementation passes the test case.

4.2 Coverage-based Test Purposes

Based on a given coverage criterion we automatically generate test properties and regular expressions specifying a test purpose satisfying the coverage criterion.

With the domain's boundary values a and b we can formulate a test purpose $([\hat{}a]*a[\hat{}b]*b)|([\hat{}b]*b[\hat{}a]*a)$ ensuring domain coverage. It reads as follows: start from any value in the domain but a boundary value, then reach one of the boundary values, and finally the other one. Such a test purpose covers all domain values of a certain quantity. We generate test purposes and furthermore test cases for all input and output quantities of the test model (relevant quantities).

We can describe delta coverage with the same regular expression as for domain coverage, but with fixed boundary values min and $plus$ since $\delta \in \{min, zero, plus\}$. In order to express complete delta coverage we generate test purposes for each possible pair of domain value and delta, e.g., (value1, min), (value1, zero), (value1, plus), ..., resulting in $\{3 \cdot card(qs(q))|q \in Q\}$ combinations. Such a test purpose looks like $[\hat{}p] * p$, where p is the property to be searched for.

For state coverage we create test purposes like for complete delta coverage but the properties we search for are complete quantity assignments corresponding to a certain state. We generate test purposes for every state in the specification. To ensure transition coverage we use a test purpose for every transition in the specification, e.g., $. * pq$. Here p specifies the start state and q the end state of the transition to be covered.

4.3 Demonstration and Results

The QR TS resulting from simulation of the two-tank example system introduced in Section 2 comprises 113 states with 305 transitions. In order to evaluate our proposed coverage criteria we use transition coverage of obtained test suites on the specification as reference measure. Tables 1 and 2 list from left to right the coverage criterion, the number of generated test purposes *#TPs*, the number of obtained test purposes *#TCs*, and the transitions coverage on the specification. For Table 1 we exhaustively extract test cases from CTGs until all transitions of the according CTG have been considered. Table 2 lists each criterion for only one generated test case per test purpose.

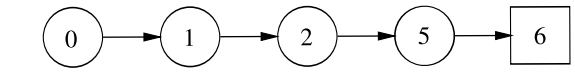
Table 1. All Test Cases per Test Purpose

Coverage Criterion	#TPs	#TCs	Transition Cov.
Domain	5	120	220/305
Delta	5	133	217/305
Complete Delta	60	850	297/305
State	113	1434	305/305
Transition	305	2256	305/305

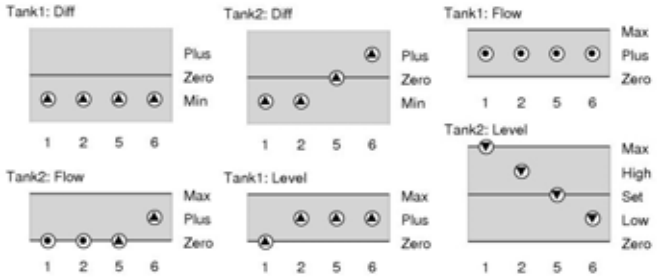
Table 2. One Test Case per Test Purpose

Coverage Criterion	#TPs	#TCs	Transition Cov.
Domain	5	5	35/305
Delta	5	5	60/305
Complete Delta	60	42	119/305
State	113	113	168/305
Transition	305	305	305/305

Figure 3 depicts an example test case created for domain coverage on quantity *Tank2:Diff*. Figure 3(a) shows a state sequence leading to an accept state and Figure 3(b) shows the according value history. With regard to delta coverage this test case covers $\delta = zero$ and $\delta = plus$ for quantity $Tank2 : Flow$. The test case also covers 3 out of 8 possible value/δ combinations for complete delta coverage, and 5 states and 4 transitions.



(a) Test Case



(b) Value History

Figure 3. Test Case for Domain Coverage for Quantity Tank2:Diff.

5 Related research

Tretmans [9] described test case generation for Labeled Transition Systems (LTS). The paper focused on Input-Output-LTS (IOLTS) and introduced conformance relations for them. The proposed testing theory also deals with states

where quiescence is allowed. Jard and Jeron [7] presented a tool for automatic conformance test generation from formal specifications. They used IOLTS as formal models and defined the *ioco* conformance relation for weak input enabled systems. Test cases are generated using defined test purposes.

Auguston et al. [1] introduced the use of attributed event grammars for generating test-cases from environment models for reactive systems. In the paper the authors use the grammar for representing an event-based model. Possible execution traces of the model form the test-cases. Insofar the underlying idea for test-case generation as described in this paper is very similar, but can be distinguished with respect to the underlying modeling language. Whereas Auguston et al. are using attributed event grammars, in this paper we are proposing the use of qualitative models for test-case generation.

6 Conclusions

In this paper we presented a set of coverage criteria based on QR models: Domain coverage, delta coverage, and complete delta coverage, as well as state and transition coverage for QR TS resulting from simulation of QR models. In contrast to previous work, these coverage criteria can be used to measure how thoroughly testing is done with regard to a model that includes an qualitative environmental description. Qualitative models represent all possible physical behaviors of systems and their environments, and can be used to find test cases which might not be considered when only using a system's specification. The approach is especially well suited when a physical model is available, e.g., in the embedded systems area. We described a method to automatically derive test cases for the proposed criteria, and illustrated the feasibility on a case study.

Initial results on a case study application show that useful test cases can automatically be generated from QR models using the described coverage criteria. As expected, the experiments show that the different criteria can be used to vary the amount of test cases generated from a QR model. We are currently in the process of evaluating the approach on models derived from Matlab Simulink models via qualitative abstraction. First experiments indicate a sound test case execution, and resulting test cases exercise the interactions of a system under test with its environment. Future work will include application of the presented methods to larger models.

Acknowledgements This work has been supported by the FIT-IT research project *Self Properties in Autonomous Systems(SEPIAS)* which is funded by BMVIT and the FFG, and the EU project ICT-216679, Model-based Generation of Tests for Dependable Embedded Systems (MOGENTES). The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

References

[1] M. Auguston, J. B. Michael, and M.-T. Shing. Environment behavior models for scenario generation and testing automation. In *International Workshop on Advances in Model Based Testing (A-MOST 2005)*, St. Louis, Missouri, USA, 2005. ACM.

[2] A. Bouwer, J. Liem, and B. Bredeweg. *User Manual for Single-User Version of QR Workbench*. Naturnet-Redime, STREP project co-funded by the European Commission within the Sixth Framework Programme (2002-2006), 2005. Project no. 004074. Project deliverable D4.2.1.

[3] H. Brandl, G. Fraser, and F. Wotawa. Qr-model based testing. In *AST 2008, Third Workshop on Automation of Software Test*, 2008. To appear.

[4] B. Bredeweg, A. Bouwer, J. Jellema, D. Bertels, F. F. Linnebank, and J. Liem. Garp3 - a new workbench for qualitative reasoning and modelling. In *Proceedings of 20th International Workshop on Qualitative Reasoning (QR-06)*, pages 21–28, Hannover, New Hampshire, USA, 2006.

[5] B. Bredeweg, J. Liem, A. Bouwer, and P. Salles. *Curriculum for learning about QR modelling*. Naturnet-Redime, STREP project co-funded by the European Commission within the Sixth Framework Programme (2002-2006), 2005. Project no. 004074. Project deliverable D6.9.1.

[6] K. D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[7] C. Jard and T. Jeron. TGV: theory, principles and algorithms. *International Journal on Software Tools for Technology Transfer (STTT)*, 7(4):297–315, October 2004.

[8] B. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289–388, 1986.

[9] J. Tretmans. Test generation with inputs, outputs, and quiescence. In *TACAS '96: Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 127–146. Springer-Verlag, 1996.

Traceability Models to Control an Aspectual Model -Driven Development

Marta S. Tabares
PI/Escuela de Ingeniería de Antioquia, Colombia
pfmstabare@eia.edu.co

Raquel Anaya
Escuela de Ingeniería, DS/Universidad EAFIT Colombia
ranaya@eafit.edu.co

Ana Moreira
CITI/Departamento de Informática, FCT/Universidade Nova de Lisboa, Portugal
amm@di.unl.edu.pt

João Araújo
CITI/Departamento de Informática, FCT/Universidade Nova de Lisboa, Portugal
ja@di.unl.edu.pt

Fernando Arango
Escuela de Sistemas Universidad Nacional de Colombia, Colombia
farango@unalmed.edu.co

Abstract

Model-Driven Development and Aspect-Oriented Software Development are paradigms that provide mechanisms to support the requirement evolution. While model-driven development contributes with a standard way to make model transformations through different abstraction levels, aspect-oriented software development provides mechanisms to separate concerns and compose crosscutting concerns. In this paper, we present Traceability Models as patterns to control the model transformations during the architectural separation of concerns. Thus, we can achieve quality features in the development process such as decreasing conflicts between stakeholders, avoiding invasive changes, and wrapping traceability in the modeling tasks.

1. Introduction

Controlling software requirement evolution is one of the most important activities in the development process to achieve reliable software products. Although several traceability approaches have been proposed, this topic is still under research [1-4]. In our investigation, we have found some issues that could frustrate a good traceability practice: (1) the software development process complexity; (2) requirements evolve, but the corresponding model elements are not changed accordingly in remaining abstraction levels; (3) the changes might be invasive (tangled or scattered); (4) development models change, but support artifacts such as traceability matrices are not updated. Moreover, in the software industry, traceability is carried out as an additional task to the modeling tasks, and its use and maintenance depends on quality standards and model information adopted by the developers.

Nevertheless, new development paradigms could provide model elements and development strategies that might help to resolve traceability issues during requirement evolution.

Model-Driven Software Development (MDSD) provides transformation mechanisms based on patterns that provide rules to transform models in different abstraction levels, and thus to preserve the architectural separation of concerns. But, the model complexity and the transformation of relationships are not dealt with. On the other hand, traceability is commonly understood as the information retrieved from transformation operations [5-7].

Aspect-Oriented Software Development (AOSD) provides mechanisms to promote separation of concerns throughout the software lifecycle, by identifying, modularizing and later composing crosscutting concerns. These features might help with the control of the concern traceability [8-12].

In our approach, we use MDSD and AOSD to control requirement evolution. Thus, we define traceability as the skill to control transformation of concerns and change propagation in different abstraction levels, maintaining decomposition and composition of crosscutting concerns by means of traceability models. This approach considers the following objectives:

1. To establish traceability models to control the concern transformations and propagate changes in different abstraction levels.
2. To deal with the decomposition of concerns and composition of crosscutting concerns through the architectural separation of concerns.

The structure of this paper is as follows. Section 2 defines traceability models. Section 3 establishes traceable model elements. Section 4 defines the concern roles. Section 5 presents related works and compares them with our approach. Finally, Section 6 concludes and shows directions for future work.

2. Traceability Models

Traceability models are instances of the metamodel shown in Figure 1. A Traceability Model is a pattern to control the model transformations, support the concerns evolution, and help in complementary tracing tasks such as the change cost-benefit analysis, and verification of consistency and completeness of system models. Each traceability model is composed by traceable model elements defined in a meta concern space where they can be identified by means of following roles: axis of tracing, predecessor, and successor. In order to achieve concern transformations, the concerns that act as axis of tracing are the *sourceElement* of the transformation and have associated a set of rules (*TransformationRule* class) to determine predecessors and successor elements (*targetElements*). They control the transformation and change propagation of concerns by means of the *TraceabilityModel – Engine*, which is supported by two classes: V*ersionController* and *PropagationController*.

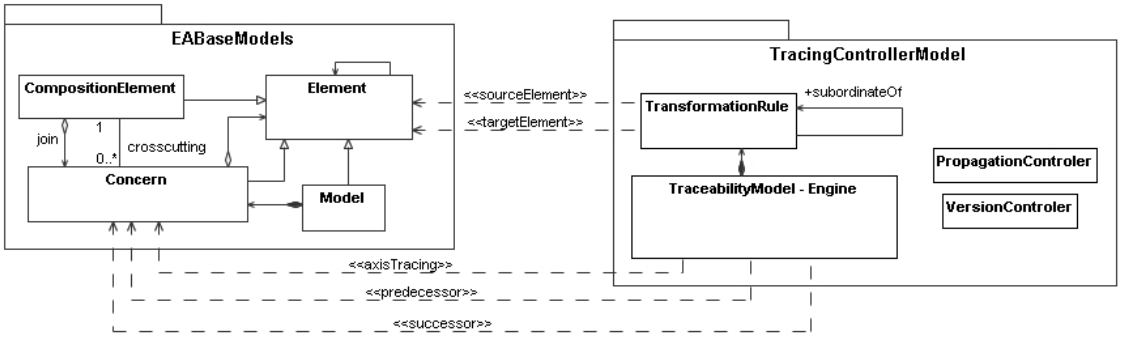


Figure 1. Meta Model for Traceability

3. Traceable Model Elements

A traceable model element is a *concern* which we see as a coherent collection of model elements, e.g., requirements, use cases, classes, collaborations, etc; this describes or represent a specific objective or function in the system. Each concern is represented by a UML package that uses a specific stereotype to identify the type of behavior or information grouped in it.

In order to reduce the complexity and provide a standard modeling for traceability practice, we make the following separation of concerns:

<<*Functional Concern*>>: this concern gathers model elements that represent a basic function of a system.

<<*QualityService Concern*>>: this concern gathers model elements that represent quality services for the system. For example: usability, security, persistency, etc.

<<*BusinessRules Concerns*>>: this concern gathers model elements that represent either functional constraints or business rules. For example: taxes, discounts, statistics, etc.

<<*Information Concern*>>: this concern gathers model elements that represent requirements or entities of information.

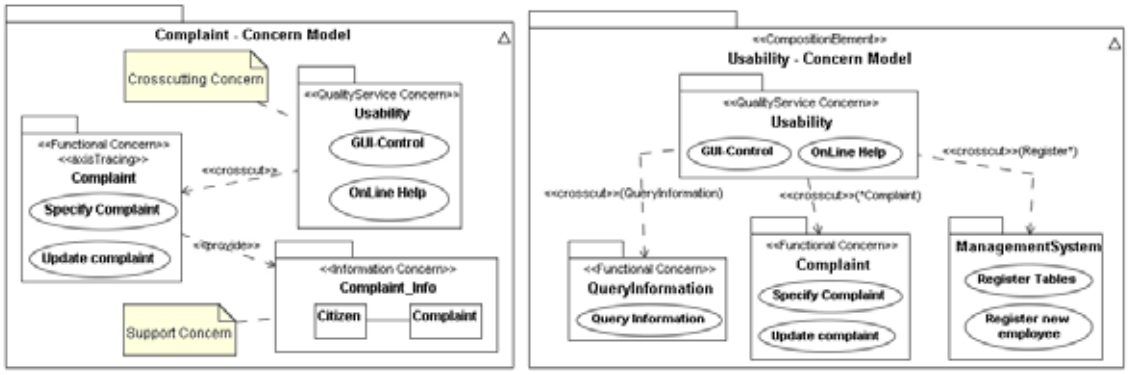
<<*Context Concern*>>: this concern gathers model elements that represent internal or external subjects or entities that generate input or output events.

Concerns such as Information, and Context are called *Support Concerns* since they help to achieve the objectives of Functional Concerns. Between these concerns, we use relationships such as <<provide>> or <<use>>. Concerns such as Quality Service, and Business Rules are called *Crosscutting Concerns* because they crosscut with additional functionality several Functional Concerns (they are considered *Support Concerns* when they only contribute with one Functional concern). Specifically, the <<crosscut>> relationship establishes the

composition between a crosscutting concern and Functional concerns.

Furthermore, an architectural separation of concerns is achieved by means of *concern models*. They are a set of interrelated concerns that represent the decomposition and composition of a problem in an abstraction level (e.g., requirements or architectural levels). Relationships between concerns are stereotypes (e.g., <<provide>>, <<crosscut>>) defined according to the types of concerns related.

In order to illustrate our ideas, we use the case study that deals with the Health Watcher System (HWS) case study. The HWS is a web-based application to manage health complaints [13]. Several concerns and their model elements have been identified, but due to lack of space we will concentrate on the concerns *Complaint*, and *Usability*. Figure 2(a) shows a view of the <<Functional Concern>> *Complaint* which is supported with the <<Information Concern>> *Complaint-Info* and crosscut by the <<Quality Service Concern>> *Usability*. Figure 2(b) shows the <<QualityService Concern>> *Usability* crosscutting the <<Functional Concern>>: *Login*, *QueryInformation*, *Complaint*, and *ManagementSystem*. Both views are constructed in the requirement level.



(a) **(b)**

Figure 2. Two Requirements Concern Models views, for the Health Watcher System.

4. Concern Roles in the Traceability Model

Traceable model elements play three basic roles in traceability models:

axisTracing (C, {r_i}). A kind of concern ***C*** is declared as an axis of tracing jointly with its transformation rules (r_i). An *axisTracing* concern controls transformations, change operations and their propagation.

Predecessor. A predecessor concern C_{Pr} precedes an *axisTracing* concern that has matched backward transformation rules to generate it.

Successors. A successor concern C_{Sc} succeeds the *axisTracing* concern that has matched forward transformation rules to generate it.

The tracing links between them are realization <<realize>> and refinement <<refine>> UML relationships:

<<realize>>(source, target). Trace relationship between *axisTracing* and *successor* tracing elements. The source is the *successor*, and the target is the *axisTracing*. Commonly, they are in different abstraction level, and *target* elements realize *source* elements.

<<refine>>(source, target). Trace relationship between *axisTracing* and *predecessor* tracing elements. The source is the *axisTracing*, and target is the *predecessor*. They can be in the same abstraction level, and source elements refine target elements.

In order to avoid invasive changes in the system and verify easier consistency and completeness quality attributes in the models, we define two kinds of traceability models:

(i) *Centered in Functional Concerns*: the *axisTracing* concern is a Functional Concern that gather use cases (see Figure 3).

(ii) *Centered in Composition Elements*: the *axisTracing* concern is a model element identified as crosscutting concern and is part of a <<CompositionElement>>.

Other ones might be defined by developers according to necessity of tracing and evolution of model elements.

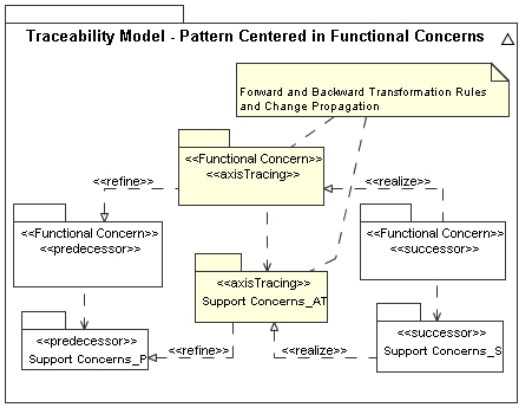


Figure 3. Pattern Centered in Functional Concerns.

Figure 4 shows a first version of a traceability model centered in Functional concerns for the *Complaint* concern. <<Functional Concern>> Complaint and <<Information Concern>> Complaint_Info have been declared as <<axisTracing>> concerns. The first gathers both Specify Complaint and Update Complaint use cases as model elements that represent the complaint function. The second (support concern) helps to make the complaint actions and gathers the Complaint class. Each one has a set of transformation rules to generate predecessors and successors concerns. In this example we consider concern models in two abstraction levels: *Requirements Concern Model (RCM)* whose instances are created at the requirement level, and their concerns gather model elements such as requirements, use cases and entities; *Architectural Concern Model (ACM)* whose instances are created (by means of transformation rules) at the architectural level, and their concerns gather model elements such as use case realizations, classes, operations, etc.

Each transformation rule is unique and consistent to guarantee the homogeneity of the concerns in different abstraction levels. Two types of rules are defined: *Root rules*, which transform concerns and their relationships, and *Subordinate rules* which transform model elements of the concerns. Each rule is composed by source and target elements. The transformation is bidirectional (forward and backward), horizontal (between concerns of the same abstraction level), and vertical (between concerns of different abstraction levels).

Table 1 show some rules performed from the *axisTracing* elements where the source concern model is in requirement level, and the target concern model is in architectural level (i.e., analysis architecture).

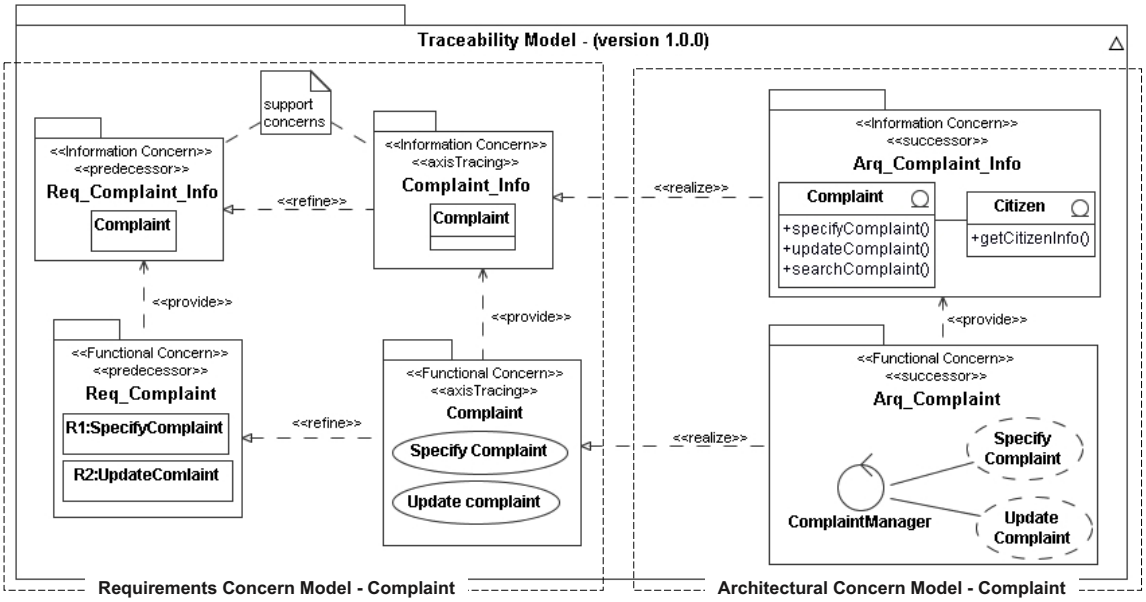


Figure 4. Traceability Model for the Complaint Concern.

Table 1. An Example of Transformation Rules.

Rule (*source2target*)	Description
Root rule #1: Req_FunctionalConcern2Arq_FuctionalConcern (Forward, vertical)	A Function concern from RCM (e.g., Complaint) is transformed into a Functional concern in the ACM.
Subordinate rule #1.1: UseCase2RealizationUC+ControlClass+Operation	A base use case (e.g., Specify Complaint) of a Functional Concern (e.g., Complaint) in the RCM is transformed into the following target model elements in the ACM: use case realization, a control class, its operation, and the association relationship with the entity class in the Information Concern of the ACM (transformed by other rule). ACM model is illustrated in the Figure 4.
Root rule #2: Req_InformationConcern2Arq_InformationConcern (Forward, vertical)	A Information concern from RCM is transformed into an Information concern in the ACM; e.g.,
Subordinate rule #2.1: Entity2EntityClass+Ops	An Entity (e.g., Complaint) of an Information Concern in the RCM is transformed into an analysis Entity Class and their basic operations, all as target model elements in the ACM.

Although these rules are written in a natural language, also they might be written in some formal transformation language.

Transformation rules have to be very well defined to make accurate change propagation. Thus, since <<axisTracing>> elements the developer can measure the change impact and execute a sequence of chained rules defined according to relationships among concerns.

5. Related Work

Almeida et al. provide a methodological framework that allows designers to relate requirements to the different design process products driven to models. This framework is a base used to trace requirements and quality evaluation of the model transformation specifications for metamodels, models and realizations. The traceability is presented in a cross reference table between requirements and models on different abstraction levels [14].

Berg et al. define crosscutting concerns based on a traceability pattern; besides, the impact analysis is based on traceability of dependencies between elements in software artifacts in both metamodel and model [15].

Kurtev et al. tackle analysis of change in the model transformation, generating traces between source and target model elements. Each trace is formed by sets of source and target elements and a rule that uses theses elements. The trace model is generated after the execution of a transformation [16]. In our approach, traceability models are patterns to control the concerns transformation and change propagation. These models help to assess the change impact analysis by means of a change management method.

6. Conclusions and Future Work

Our approach incorporates traceability as part of the development process, defining traceability models as patterns to control concerns evolution based on architectural separation of concerns, and concern transformations.

Architectural separation of concerns by means of concern models allows us to trace type of concerns, preserving their decomposition and composition in different abstraction levels. In other words, we achieve horizontal and vertical separation of concerns from requirement to architecture using stereotyped packages and concern models. In addition, crosscutting concerns can be traced independently, without losing the consistency with the basic models, since we define horizontal and vertical composition of concerns and its transformations. These features help to diminish the complexity in the development process, avoid invasive changes, and resolve conflicts between stakeholders from requirement specification.

In traceability models, the transformation is controlled by means of concerns that act as axis of tracing. They have a set of root and subordinate rules which guarantee concerns homogeneity from requirements to architecture, and propagate the change to predecessor and successor concerns. These models are used in a method to support the changes management. Hence, both models and matrices of traceability are instances of them and their update is automatic.

We are currently developing a traceability method based on the traceability models in an academic modeling CASE tool.

7. References

[1] O. Gotel, and A. Finkelstein, "Extended requirements traceability: results of an industrial case study". 3rd IEEE Intl. Symposium on Requirements Engineering (RE'97).

[2] B. Ramesh, and M. Jarke, "Towards reference models for requirements traceability". IEEE TSE, 27(1), 2001.

[3] A. Egyed, "A scenario-driven approach to trace dependency analysis". IEEE TSE 29(2): 116-132, 2003.

[4] J. Cleland-Huang, C.K. Chang, and M. Christensen, "Event-based traceability for managing evolutionary change," TSE 29(9), Sept. 2003 Page(s):796 – 810.

[5] MDA-Guide, *OMG Document v1.0.1*, omg/03-06-01.

[6] S.J. Mellor, A.N. Clark, T. Futagami, "Guest Editors' Introduction: Model-Driven Development". IEEE Software 20(5): 14-18 (2003).

[7] Stahl, T., and M. Volter, *Model-Driven Software Development-Technology, Engineering, Management*, John Wiley and Sons, Ltd., Chichester, England 2006. (2005).

[8] Filman, R.E., T. Elrad, S. Clarke, and M. Aksit, *Aspect-Oriented Software Development*. Addison Wesley 2004.

[9] A. Moreira, A. Rashid, and J. Araujo, "Multidimensional Separation of Concerns in Requirementes Engineering", Intl. Conf. on RE, 2005.

[10] Jacobson, I., P.W. Ng, *Aspect-Oriented Software Development with Use Cases*, Addison Wesley Professional (2005).

[11] Clarke, S., E. Baniassad: *Aspect-Oriented Analysis and Design. The Theme Approach*. Ed: Addison-Wesley, Object Technology Series (2005).

[12] R. Chitchyan, A. Rashid, P. Sawyer, et al., "Survey of Aspect-Oriented Analysis and Design Approaches", AOSD-Europe-ULANC-9, May 2005.

[13] Health Watcher Case Study, http://aosd.di.fct.unl.pt/ea-icse2007/documents/Health_Watcher_Usecase_v2_1.pdf.

[14] J.P. Almeida, P.Eck, M.E. Iacob, "Requirements Traceability and Transformation Conformance in Model-Driven Development", Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06).

[15] K. van den. Berg, B. Tekinerdogan, and H. Nguyen, "Analysis of Crosscutting in Model Transformations", ECMDA Traceability Workshop. 2006.

[16] I. Kurtev, M. Dee, A. Goknil, K. van den Berg, "Traceability-based Change Management in Operational Mappings", ECMDA Traceability Workshop 2007.

Knowledge-based system development with scripting technology – a recommender system example

Dietmar Jannach
Department of Computer Science
Dortmund University of Technology, Germany
e-mail: dietmar.jannach@udo.edu

Abstract

The core functionality of many knowledge-based systems is built with the help of special-purpose software components and programming environments such as rule engines or Prolog interpreters. Other parts of the application – like the Web interface – are however built with "standard" software development technology like Java which means that not only the corresponding interfaces and data exchange mechanisms between the components have to be developed, but also that the software developers have to work with different technologies or even programming paradigms.
In this paper we show how recent "scripting" extensions in a programming language like Java can be exploited to develop highly flexible and extensible knowledge-intensive applications. The different advantages of such an approach are discussed based on the experiences gained from developing a scripting-based software library for building knowledge-based recommender applications.

1. Introduction

The possible advantages of a knowledge-based software development approach are well known. The domain knowledge can be separated from the reasoning knowledge while at the same time optimized off-the-shelf reasoning engines can be used to automate the inferencing or problem solving process.

In many real-world or commercial applications, however, only a part of the "intelligent" system will be developed with the help of special programming environments such as LISP or Prolog or by use of a rule-engine like Jess[1]. The web interface or the database layer will most probably be developed with the help of "standard" technology such as Java, Servlets, or Java Server Pages. This mix of technologies brings additional complexity to the software development process since programmers not only have to implement the different interfaces and data exchange channels but – more importantly – are faced with different programming paradigms.

In the field of programming languages, we can observe signs of a revival of "scripting" languages in recent years. Despite their disadvantages with respect to type-safety, compile-time problem detection and run-time performance, languages like PHP, Python and recently Ruby became popular in particular for web application development as they – according for instance to [13] – promise to be advantageous with respect to flexibility and developer productivity. From our point of view, the aspects of flexibility and in particular extensibility that these interpreted languages provide also make them interesting for the development of knowledge-intensive systems.

In this paper, we report and discuss our experiences gained from developing the light-weight JPFINDER library for building knowledge-based recommender systems . The library is fully written in the Java programming language and exploits the language's recent "scripting" support to achieve the required levels of extensibility and compactness of the knowledge bases. Overall, the work shall thus exemplify one of the options of embedding knowledge-based system development into industrial software development processes and environments.

2. Application domain & system architecture

In the field of recommender systems, the following main approaches can be distinguished: Classical *community-based recommender* systems base their proposals on item ratings, user similarities, and collaborative filtering techniques, see [1] for a recent overview. *Content-* or *knowledge-based* systems on the other hand operate on the basis of further pieces of information, which can for in-

[1] http://www.jessrules.com/jess/index.shtml

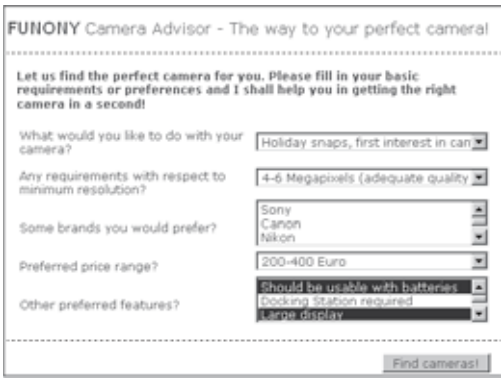


Figure 1. Preference elicitation

Figure 2. Recommendation & explanation

stance be knowledge about the items to be recommended and/or knowledge about directly or indirectly acquired preferences of the users. Typical techniques used in the latter type of systems are for instance filter-based matching, similarity-based retrieval or utility-based ranking [4].

Screenshots of a typical interactive recommender application in the sense of [5] are shown in Figure 1 and Figure 2. The user is first guided through a series of possibly personalized questions in order to determine her specific needs (Figure 1). At the end of this dialog, the system comes up with a recommendation and – due to its knowledge-based nature — is also capable of explaining the reasons for this particular proposal, retrieve other similar catalog items, or let the user revise or specify additional requirements.

It can be easily observed that such applications are *knowledge-intensive*, meaning that various pieces of domain-specific information have to be encoded in the background. Examples for such knowledge chunks are for instance the *matching rules* that determine which items suit some given requirements, utility functions for ranking the items, or personalization rules to adapt the user interface and navigation path according to the current user profile.

In [5], CWADVISOR, an integrated environment for the development of such knowledge-based and highly interactive recommender systems is presented. The CWADVISOR system is designed as a classical and to some extent heavy-weight expert system. It relies on the explicit representation of domain knowledge, provides a fully-fledged graphical knowledge acquisition component, a relational database for persisting the knowledge as well as proprietary languages for modeling filtering or personalization rules.

The JPFINDER library discussed in this paper implements many of the features of CWADVISOR and was designed to be a light-weight alternative to the comprehensive CWADVISOR expert system. In particular it should also take advantage of recent scripting features of the Java programming language as to reduce the development efforts that are required for rule processing while at the same time extensibility and flexibility should be retained.

In the work presented herein, we will particularly focus on how the *rule knowledge* is represented and processed, as the development of software systems that are governed by *business rules* are in wide-spread industrial use also in other application domains. In the CWADVISOR system, a proprietary rule language is used. The statements are either parsed, translated and compiled to Java code [8] or directly executed with the help of a proprietary rule interpreter [5]. In either case, a comparably complex parser and compiler component as well as a graphical tool for modeling the rules (including for instance auto-completion and correction features) are required, whereas JPFINDER relies on scripting technology for that purpose.

The overall architecture of the JPFINDER system is outlined in Figure 3. At the core, the *Recommender Engine* interacts with the different users of the system and correspondingly maintains *Recommender Session* objects that contain the current user's profile. The *Engine* has a defined Application Programming Interface to manipulate the domain knowledge which can also be used for loading the definitions from a persistent data store at system startup. Several pluggable *reasoning modules* are also part of the library and implement specific functionalities such as filter-based matching, utility calculation or techniques for user interface personalization.

3. Recommendation technique implementation

In the following, we will discuss the logic of JPFINDER and some of its modules in more detail and in particular focus on the scripting-based implementation of the functionalities.

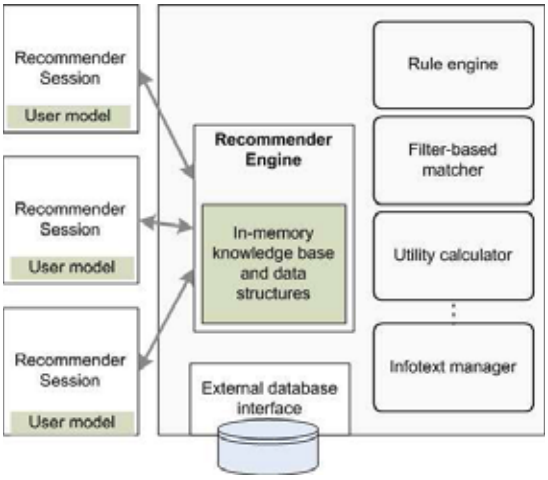


Figure 3. Architecture overview.

3.1. The user & product model

In JPFINDER, every piece of information about the user and the preferences used for generating recommendations are contained in the *user model*. In order not to limit the generality of the approach, the user model is thus generally defined to be a finite set of variables *UV* (user variables), each of them with a defined data type. Variables can be either string-valued or numeric; of both types, multi-value instantiations are possible, see for instance the choice of preferred features in Figure 1. Note again that the values for variables in *UV* do not necessarily have to be directly acquired through questioning but can also be derived "internally", e.g., based on scoring schemes or other value derivation rules. The set of specific (input) values for one individual user shall be denoted as *IV*. In fact, also complex user modelling techniques – that for instance maintain a history of user interactions – can be employed. The implementation of such profiling or learning techniques is however beyond the scope of JPFINDER.

Similarly, the *product model* defines the characteristics of the items to be recommended. The product characteristics are described by a set of variables *PV* (product variables) with defined data types. Beside regular product features, the product model can also be used to describe "external" product characteristics such as vendor reliability or current stock availability in a given shop.

Each individual recommendable item is thus described by a set of key-value pairs; the set of all items forms the product is denoted as product catalog *PC*.

In the example, *UV*, *PV*, *PC* and some user inputs IV_1 could be as follows, using Java notation for data types.

$$UV = \{pref\_price : Double, \\ pref\_features : String[], ..\}$$

$$PV = \{price : Double, resolution : Double, ..\}$$
$$PC = \{\{price : 400.00, resolution : 3, ...\}, \\ \{price : 300.00, resolution : 2, ...\}\}$$
$$IV_1 = \{pref\_price = 100, pref\_features = ..\}$$

3.2. Filter-based matching

Model. With *filter-based matching* we mean a technique in which customer preferences are directly or indirectly mapped to constraints on product properties. Such filter-based approaches are also commonly used in similarity-based systems to pre-filter the set of products to be compared [11].

In JPFINDER, the mapping from user preferences to desired product characteristics can be expressed in the form of "if-then-style" rules, i.e., a filter rule $FR < AC, FC >$ consists of an *activation condition AC* and a *filter constraint FC*. We use $FR.AC(IV)$ to refer to the evaluation of the activation condition given input values IV and $FR.FC$ to refer to the filter constraint definition.

A typical filter rule in the domain could be: *"**If** the user prefers the lower price range **then** recommend items with a price lower than 200 Euro."* Formally, filter rules are interpreted as follows. Let AC be an expression over variables in UV, FC an expression over the variable set $UV \cup PV$. IV are the customer input values and PC is the product catalog given in the form described above.

Given these inputs, the "catalog query" Q is thus the conjunction of FC-expressions of those filter rules for which AC evaluates to *true*, i.e.

$$Q \equiv \bigwedge(f.FC) | f \in FR \wedge f.AC(IV) = true$$

If we interpret the product catalog PC as a relational database table, filtering the set of recommended products RP corresponds to performing the database query σ_Q on PC.

Scripting-based implementation The implementation of filter-based matching in JPFINDER relies on a recent algorithm [9] which also supports fast query relaxation for failing queries and which is based on compact in-memory data structures and partial in-advance query evaluation.

Note that although there have been several efforts to defining a common knowledge interchange format (see, e.g., KIF[2]), no common representation mechanism for rules or constraints is yet broadly established. In more recent efforts, the general constraint language ESSENCE [7] has been proposed and SWRL[3] has been submitted to the W3C as a rule language in the Semantic Web. Still, besides problems of syntactic complexity (SWRL) or limited expressiveness (ESSENCE), special purpose parsers, compilers, or interpreters have to be employed to support these languages.

[2] http://logic.stanford.edu/kif/kif.html
[3] http://www.w3.org/Submission/SWRL/

```
<xml>
  <FilterRule>
    <if condition="c_pref_price_range = 'medium'"/>
    <then constraint="p_price > 200 && p_price < 400"/>
    <explanation text="I could obey your price preferences
                       (medium price range)"/>
    <excuse      text="Note that due to your other requirements I could
                       not obey your price preferences"/>
    <priority="3"/>
  </FilterRule>
  <FilterRule>
     <if condition="..."/>
                ...
</xml>
```

Figure 4. XML representation of filter rules.

As a knowledge representation mechanism for the filter constraints, JPFINDER therefore relies on JavaScript, which is the default scripting language supported by the recent Java 6 SE release. As described above, the filter rules are written in a simple "if-then"-style, which in our experience (see, e.g., [5]) is easy to comprehend also for domain experts who are not experienced in programming or specific knowledge representation techniques. The activation condition and the filter constraint are formulated as JavaScript expressions over the variables of the user model.

In Figure 4, an example of a filter rule including relaxation priorities and explanation texts in the sense of [9] is shown. At run time, the variables of the user model – which may have been acquired via an ordinary Java Server page or from a more elaborate user modelling component – are forwarded to the scripting engine which then evaluates the activation conditions of the rules. Those expressions that evaluate to true are then used to filter the suitable catalogue items. Note that The XML representation in Figure 4 is optional as the library provides an appropriate Java-based application programming interface for registering the filter rules.

Within the expressions statements, all JavaScript language constructs can be used and arbitrarily complex calculations are thus possible. The actual values of the variables of the user- and product model are automatically put into the scope of the scripting engine. With respect to extensibility aspects, the chosen scripting approach also allows us to define custom JavaScript functions that can be seamlessly used within expressions. Consider, for instance, that a filter constraint should be written that is activated whenever the user has chosen a particular value from a *set of options* like *"**if** the customer preferences (among others) contain 'usb' **then** ..."*.

To that purpose, the knowledge engineer can write a standard JavaScript program that tests for set membership:

```
function isContainedIn(value,um_var) {..}
```

Custom functions like this can then be registered to the recommender engine at runtime and used within the filter constraint. Internally, the script code are automatically compiled into Java code for performance reasons, which is a standard feature of Java SE. The only task of the recommender engine is to put the current values of the session into the execution scope when the function is called.

3.3. Utility- and similarity-based retrieval

Retrieving items based on their expected utility to the user or based on the similarity with (individual features of) another item are two other techniques used in knowledge-based recommendation, see [2] or [3].

At the core of *utility-based approaches*, a *utility function* is used whose value either depends on specific product features alone or which also takes the user's preferences into account. A typical evaluation scheme for determining such personalized utility values can for instance be based on Multi Attribute Utility Theory (MAUT) [8, 14].

Within JPFINDER, such arbitrarily complex utility functions can be defined in a similar way like the custom extensions mentioned above, i.e., with the help of JavaScript functions.

```
<UtilityFunction type="dynamic">
      function myUtility() {
            var utility_total = 0;
            // Utility dimension mobility
            var utiltiy_mobility = 0;
            if (p_weight < 300)
                  utiltiy_mobility += 3;
            if (p_batteries = 'yes')
                  utiltiy_mobility += 2;
            ...
            // Weighting based on user prefrences
            if (pref_mobility == 'high')
                  ...
            return utility_total;
      }
</UtilityFunction>
```

Figure 5. Fragment of utility function.

A fragment of a possible utility function that both evaluates product features and user preferences is sketched in Figure 5. At run time, JPFINDER applies the function on

each catalog item separately. The resulting utility values can then be used to sort the items in the recommendation list accordingly. JPFINDER supports two variants of utility functions, *static* and *dynamic* ones. If a utility function is marked to be static then no user model variables must be used in the function. This differentiation is mainly introduced for performance purposes. If no user model variables are used, the values of the utility function will be the same for all customers, which in turn means that the corresponding values can be pre-computed when the library is initialized. "Dynamic" evaluation functions have to be evaluated in the context of the requirements of a specific user like in the MAUT approach.

In the same way, custom functions can be developed to implement *similarity-based retrieval* recommendation techniques. In contrast to utility functions, the return value of a similarity function is not an individual utility value but rather a normalized numerical value that describes the relative similarity between two items. Based on this mechanism, JPFINDER thus supports the implementation of recommender systems like the Wasabi personal shopper [3] or of critiquing approaches [10], in which the user can ask the system to retrieve items that are similar to a given one with respect to some features.

3.4. Rule-based inferencing

In some application domains, additional forms of inferencing (on user model variables) are required. This could be for instance the "internal" derivation of further variable values based on business rules or the determination and personalization of dialog pages in an interactive preference elicitation process [5].

JPFINDER supports the implementation of business rules or additional personalization logic through generic extension patterns that allow the developer to write code fragments, which are executed when certain conditions are fulfilled.

```
<BusinessRule>
  <If>
     c_pref_investment_duration > '10 years'
  </If>
  <Then>
     if (c_pref_question_1 == 'yes')
        c_derived_score += 3;
     if (...)
        …
  </Then>
</BusinessRule>
```

Figure 6. Fragment of business rule.

Figure 6 shows a fragment of a possible business rule in JPFINDER: Depending on some user input the value of *c_derived_score*, which corresponds to an internal variable of the user model, is determined. Note that the consequent of the rule can contain arbitrary code, which means it can also be used to perform mathematical calculations for, e.g., repayment rates commonly used in the financial services domain [6].

Personalized Text Fragments represent another extension pattern of JPFINDER. With the help of rules of this type, the recommender system can select personalized variants of predefined text fragments, which can for instance be used to adapt the graphical user interface according to the knowledge level of the current user. Finally, the pattern of *User Model Expressions* allows the engineer to define arbitrary expressions over user model variables. Based on the evaluation of these expressions for the current user, the dialog flow of the Web interface or other personalization features of the application can be designed in a flexible way.

The implementation of the rule execution engine in the current version of JPFINDER is a rather simple one. For the case of business rules, the engine for instance simply evaluates the rules until no more changes in the user variables can be observed. More elaborate techniques like rule-chaining or more expressive types of rules are planned for future versions.

4. Implementation aspects

Run-time performance is in general one of the key issues for interpreted scripting languages. Thus, particular attention has been paid to these aspects both in the design as well as in the implementation of JPFINDER's algorithms. What became obvious quite soon is that the "immediate" execution of text-based scripts in Mozilla's Rhino engine[4], which is the standard implementation in Java 6 SE, is not applicable for realistic problem sizes.

With respect to algorithms, let us exemplarily discuss the filter-based matching and relaxation problem from section 3.2. In particular the search for optimal "relaxations" of a failing query can be a costly operation as theoretically all combinations of subqueries have to be evaluated. In contrast to previous algorithms used for this task [11, 12], JPFINDER thus implements a novel technique [9] which is based on the in-advance evaluation of the filters and the usage of compact in-memory data structures. It was shown in [9] that this way the number of required "database queries" for determining the optimal relaxation can be limited to the number of given subqueries at the cost of slightly increased memory requirements.

Beside the usage of such recent algorithms, JPFINDER also relies on run-time "compilation" of all scripting code. When the library is initialized with the external knowledge base or additional knowledge pieces

[4] http://www.mozilla.org/rhino/

are added via the API, all JavaScript expressions and functions are automatically translated into Java byte-code. This functionality is implemented based on the built-in Java's `ClassCompiler` and dynamic class loading. Consequently, when scripting code has to be evaluated in a recommendation session, the script interpreter is actually not needed anymore as everything is already available in the form of Java byte code. Still, no manual recompilation is required when the knowledge base changes, as the needed byte code can be generated at run time.

As an example for performance numbers, consider the following rough running time numbers measured on a standard desktop computer (Intel P4, 3.2 GHz, 1 GB RAM). A recommender knowledge base for digital cameras may comprise around 400 products and 30 filtering rules, which in our experience is a realistic size. Let us assume that 15 of the filter rules are "active" in a current session and the best relaxation comprises 12 rules, i.e., three filters have to be relaxed. The running time for such a problem setting (without dynamic filter evaluation as described in [9]) is around only 100ms. Even if we double the problem size (800 products and 60 rules), the response time is still less than half a second, which is appropriate for interactive recommender sessions. Another number can be given for the dynamic construction of new filter conditions, see the "a bit cheaper models" - functionality in Figure 2. Evaluating for instance a single five-atom query with a smaller price for a catalog of 800 cameras requires less than 20ms. The overall memory requirement for both examples is below 15 megabytes, including all the product data which is kept in memory.

5. Conclusions

Based on an example from the domain of knowledge-based recommender systems, the paper has demonstrated how scripting technology can be used to simplify the development of knowledge-intensive software applications.

Compared with complex and heavy-weight expert systems that incorporate specialized programming environments or rule-processing engines, the advantage of the presented approach in particular lies in the fact that the software and knowledge engineer is not confronted with different programming paradigms and languages. In addition, the use of a consistent set of technologies simplifies the integration of such intelligent reasoning modules into standard Web development toolkits and industrial software development processes.

The preliminary evaluation of the presented library, which is based on real-world scenarios and experiences gained from previous projects [5], indicates that (1) many of the functionalities of more complex frameworks can be implemented with less code (as no special parsers, compilers, or interpreters are required), and that (2) performance issues that commonly arise in scripted languages can be successfully addressed with the help of runtime compilation.

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[2] D. Bridge. Product recommendation systems: A new direction. In R. Weber and C. Wangenheim, editors, *Workshop Programme at 4th Intl. Conference on Case-Based Reasoning*, pages 79–86, 2001.

[3] R. Burke. The wasabi personal shopper: a case-based recommender system. In *Proceedings of the 11th National Conference on Innovative Applications of Artificial Intelligence, AAAI'99*, pages 844–849, Menlo Park, CA, USA, 1999.

[4] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2005.

[5] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. An integrated environment for the development of knowledge-based recommender applications. *International Journal of Electronic Commerce*, 11(2):11–34, Winter 2006-7 2007.

[6] A. Felfernig and A. Kiener. Knowledge-based interactive selling of financial services using FSAdvisor. In *17th Innovative Applications of Artificial Intelligence Conference (IAAI)*, pages 1475–1482. AAAI Press, 2005.

[7] A. M. Frisch, M. Grum, C. Jefferson, B. M. Hernández, and I. Miguel. The design of ESSENCE: A constraint language for specifying combinatorial problems. In *International Joint Conference on Artificial Intelligence - IJCAI*, pages 80–87, Hyderabad, India, 2007.

[8] D. Jannach. Advisor suite - a knowledge-based sales advisory system. In L. S. Lopez de Mantaras, editor, 16th *European Conference on Artificial Intelligence (PAIS)*, pages 720–724. IOS Press, 2004.

[9] D. Jannach. Finding preferred query relaxations in content-based recommenders. In *IEEE Intelligent Systems Conference*, pages 355–360, Westminster, UK, 2006. IEEE Press.

[10] L. McGinty and B. Smyth. Adaptive selection: An analysis of critiquing and preference-based feedback in conversational recommender systems. *International Journal of Electronic Commerce*, 11(2):35–57, 2006.

[11] D. McSherry. Retrieval failure and recovery in recommender systems. *Artificial Intelligence Review*, 24(3/4):319–338, 2005.

[12] N. Mirzadeh, F. Ricci, and M. Bansal. Supporting user query relaxation in a recommender system. In 5th *International Conference on E-Commerce and Web Technologies (EC-Web)*, pages 31–40, Zaragoza, Spain, 2004. Springer.

[13] B. A. Tate. *From Java to Ruby*. Pragmatic Bookshelf, 1st edition, 2006.

[14] D. von Winterfeldt and W. Edwards. *Decision Analysis and Behavioral Research*. Cambridge University Press, Cambridge, UK, 1986.

Integrating Trust Management into Usage Control in P2P Multimedia Delivery

Li Yang
Dept. of Computer Science
University of Tennessee at Chattanooga,
Chattanooga TN 37415
Li-Yang@utc.edu

Raimund Ege
Dept. of Computer Science
Northern Illinois University
DeKalb, IL 60115
ege@cs.niu.edu

Abstract

Sharing files via Internet gains more and more attention now. Recent P2P protocols use a decentralized model of distributing large multimedia files that greatly alleviate the bandwidth demand on the multimedia source. Such open delivery of multimedia files demands an adaptive and robust reputation management system to facilitate file sharing, moreover, security issues such as digital rights and access control need to be resolved. Our approach handles reputation and security issues in P2P file sharing when the attributes and behaviors of the principal are uncertain and mutable. Reputation management is integrated into access control model to support decision making in P2P file sharing in which the environment is ever changing. We solve uncertainty and mutability by an adaptive and decentralized reputation system. The peer with low reputation will be separated from file sharing, and its granted on-going access will also be revoked if access control rules are no longer met. We have applied our reputation-based usage control framework to an application of P2P file sharing.
Keywords*: trust, access control, P2P*

1 Introduction

Making multimedia content available online becomes the next Killer-Application for the Internet. Services such as iTunes, YouTube, Joost. are popularizing delivery of audios and video content to anybody with a broadband internet connection. With new virtual communities emerging, users communicate directly with one another to exchange information or execute transaction in a peer-to-peer fashion. Kazaa is an example of P2P networks and bittorrent is an example of P2P protocols. These services are currently struggling with the challenges of securing large-scale distribution. The dynamism of the P2P communities means that the principal that offer services will meet requests from unrelated or unknown principals. Peers need to collaborate and obtain services within environment that are unfamiliar or even hostile. Therefore, peers have to manage the risks involved in the collaboration when prior experience and knowledge about each other are incomplete. One way to address this uncertainty is to develop and establish trust among peers. Trust can be built by either a trusted third party [1] or by community-based feedback from past experiences [15] in a self-regulating system. Trust leads naturally to a decentralized approach to security management that can tolerate partial information.

In such a complex and collaborative world, a peer can protect and benefit itself only if it can respond to new peers and enforce access control by assigning proper privileges to new peers. Access control models [4, 11] determine authorization based on principals' permission on target objects. Usage of a digital object is temporal and transient in virtual community such as on-line reading, which is beyond an instantaneous access. UCON [14] is proposed to handle continuity of access decisions and mutability of subject and object attributes. Authorization decisions are made before an access, and repeatedly checked during the access. The on-going access may be revoked if the security policies are not satisfied due to changes of the subject, object or system attributes.

The general goal of our work is therefore to investigate the design of a novel approach to addressing both uncertain information and mutable attributes. If successful, this approach will offer significant benefits in emerging applications such as P2P. It will also benefit collaboration over the existing Internet when the identities and intentions of parties are uncertain. We integrate trust evaluation with usage control to handle uncertainty of entities and mutability of attributes. Underlying our framework is a formal computational model of trust and access control that will provide a formal basis to interface authentication with authorization.

2 P2P Delivery of Multimedia

P2P delivery of multimedia aims to deliver multi-media content from a source to a client. We assume that the content comes into existence at the source, i.e. we don't want to consider storing and securing the media at the origin. Example of creating such multi media might be a video camera with microphone. Likewise the client consumes the content, e.g. by displaying it on a TV monitor. We further assume that there is just one original source, but that there are many clients that want to receive the data. Our approach is specifically geared towards being able to scale effortlessly to support millions of clients without prior notice, i.e. be able to handle a "mob-like" behavior of the clients. Some lag time between creation of source data and its consumption by clients is acceptable, but excessive wait will defeat the attractiveness of our approach.

We use a P2P approach. The source data is made available at a preset quality using a variable-rate video encoder. The source data stream is divided into fixed length sequential frames: each frame is identified by its frame number and encrypted (see next section). Clients request frames in sequence, decrypt the frame and reassemble the video stream which is then displayed using a suitable video decoder and display utility. The video stream is encoded in such a fashion that missing frames don't prevent a resulting video to be shown, but rather a video of lesser bit-rate encoding, i.e. quality, will result [18].

Multi-media sources are advertised and made available via a central tracking service: at first, this tracker only knows the network location of the server. Clients that want to access the source do so via the tracker: they contact the tracker, which will respond with the location of the source. The tracker will also remember (or track) the clients as potential new sources of the data. Subsequent client requests to the tracker are answered with all known locations of sources: the original and the known client. Clients that receive locations of sources from the tracker issue frame requests immediately to all sources. Clients will also answer requests for frames that they have received already, which will enable a cascading effect, which establishes a p2p network where each client is a peer.

Figure 1 shows an example with one source, one tracker and three clients. The source is where the video data is produced, encoded, encrypted and made available. The tracker knows the network location of the source. Tracker and source maintain a secure connection. Clients connect to the tracker first and then maintain sessions for the duration of the download: all 3 clients maintain an active connection to the tracker. The tracker informs the client which source to download from: Client 1 is fed directly from the source; client 2 joined somewhat later and is now being served from the source and client 1; client 3 joined last and is being served from client 1 and client 2. In this example, two of the clients are also serving as intermediaries on the delivery path from original source to ultimate client.

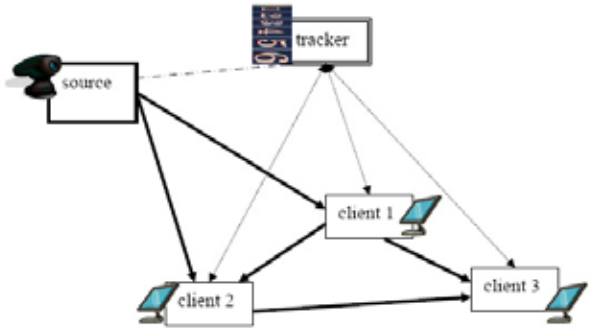


Figure 1. P2P Content Delivery Network

3 Integrating Trust into Usage Control

Integrating trust evaluation into usage control allows collaboration when attributes of a principal are mutual or information of a principal's behavior is incomplete. During collaboration, owner of resources evaluates the trust of requesting principal first and then make authorization decision of resources sharing. The trust evaluation and authorization decision are temporal since they are made before, during and after the resource sharing.

3.1 Trust Evaluation

For every request, the owner of resources assigns a trust value between 0 and 1 to the requesting principal. The trust is evaluated based on both observation and recommendations from referees. Observations are the previous interactions the owner had with the requesting principal. Recommendations may include signed trust-assertions from other principals, or a list of referees whom the owner can contact for recommendations. The owner first computes trust given a sequence of observations from interaction history, then combines the trust with recommendations. The trust value, calculated from observations and recommendations, is a value within [0, 1] interval evaluated from a principal for a request.

The trust is assumed to follow a beta distribution, and represented by the two parameters of the beta distribution. The beta distribution, a conjugate prior, is chosen because of its reproducibility property under the Bayesian framework. When a conjugate prior is multiplied with the likelihood function, it gives a posterior probability having the same functional form as the prior, thus allowing the posterior to be used as a prior in further computations. For

a given requester, we define a sequence of variables T_1, T_2,. . ., T_k characterize the trust at the sampling time k. For instance, at *kth* sampling time, N_k observations are collected by the owner. Let G_k be the number of normal requests or behaviors. If the owner did not detect attacks (i.e., spyware, Trojan horse, SQL injection) associated with a request, he/she deems the request as normal behaviors. Suppose a prior probability density function (*pdf*) of trust T_{k-1}, denoted by $f_{k-1}(t)$ is known. Then the posterior *pdf* of (given $N_k = n$ and $G_k = g$) can be obtained from Bayes theorem [13] as follows:

$$f_k(t) = \frac{f_k(G_k = g|t, N_k = n)f_{k-1}(t)}{\int_0^1 f(G_k = g|t, N_k = N)f_{k-1}(t)dt} \quad (1)$$

where $f_k(G_k = g|t, N_k = n)$ is called the likelihood function and has the form of a binomial distribution:

$$f_k(G_k = g|t, N_k = n) = \binom{n}{g} t^g (1-t)^{n-g} \quad (2)$$

The prior *pdf* $f_{k-1}(t)$ summarizes what is known about the distribution of T_{k-1}. Under the assumption that prior *pdf* $f_{k-1}(t)$ follows a beta distribution, it can be shown that the posterior pdf also follows a beta distribution.

In particular, if $f_{k-1}(t) \sim beta(\alpha_{k-1}, \beta_{k-1})$, we have $f_k(t) \sim beta(\alpha_{k-1} + g_k, \beta_{k-1} + n_k - g_k)$ given that $N_k = n_k$ and $G_k = g_k$. Therefore, $f_k(t)$ is characterized by the parameters α_k and β_k defined recursively as follows: $\alpha_k = \alpha_{k-1} + g_k$ and $\beta_k = \beta_{k-1} + n_k + g_k$. Initially, the owner has no knowledge about the requester. We assume that trust value has uniform distribution over the interval $[0, 1]$, i.e., $f_0(t) \sim U[0, 1] = beta(1, 1)$ which indicates our ignorance about the requester's behavior at time 0. At time k, trust value $\bar{t}$ of the principal is:

$$\bar{t} = \frac{\alpha_k}{\alpha_k + \beta_k} \quad (3)$$

There are two alternative ways to update trust values. One is to update trust values based on all the observations and recommendations. The other ways is to update trust values based on recent information only. The advantage of the latter one is two folds: reduce the computation complexity and detect the changing of behaviors early. For instance, a requestor is misbehaving in a short time range, then recent observation together with reports is more reflective to the behavior changing than the overall observation.

Meanwhile, recommendations from referees bring in new information T_{rq} on requester's behaviors. The owner combines the new data T_{rq} with its own observation T_{oq} on the condition that the referee is one of owner's friends or the recommendation passes the deviation test. Deviation test is to decide recommendation is trustworthy or not. Recommendation R is learned from the past interaction the referee had with the requestor. Trustworthiness of a recommendation also follows a beta distribution. $f_k(t)$ is adjusted by recommendations: $T_{oq} := T_{oq} + \mu T_{rq}$ where T_{oq} is trust that owner has to requester, T_{rq} is trust that referee has to requester, and μ is the owner's belief of referee's recommendations.

3.2 Overview of UCON

Usage control model UCON proposed by J. Park [14] is a generalization of access control to cover authorization, obligation, conditions, continuity (ongoing controls), and mutability. Authorization handles decision on user accesses to target resources. Obligations are the mandatory requirements for a subject before or during a usage exercise. Conditions are subject, object, environmental or system requirements that have to be satisfied before granting of accesses. Subject and object attributes can be mutable. Mutable attributes can be changed because of accesses, whereas immutable attributes can be changed only by administrative action.

3.3 Trust-based UCON

A state is an assignment of values to variables which consist of principal attributes, object attributes and system attributes. The state transition system can be represented by (Σ, S, s_0, δ, F) where Σ is input alphabet, S is a set of system states, s_0 is the initial state, δ is the state transition function $\delta : S \times \Sigma \rightarrow S$, and F is the final state. We define a special system state to specify the status of a single request and access process. The system state S includes $initialState$, $preTrust$,$deniedEnroll$, $trusting$, $disEnrolled$, $preAccess$, $deniedAcces$, $accessing$, $revoked$, and end. The $initialState$ means the principal has not sent request; $preTrust$ means principal is waiting for the authentication decision; $deniedEnroll$ means the system denies the enrollment of the principal based on history or recommendations; $trusting$ means the principal is allowed to collaborate and will send access requests; $disenrolled$ means the system revokes the enrollment of a principal based on runtime information. The $preAccess$ means the principal is waiting for the authorization decision; $deniedAccess$ means they system denies the authorization request based on access control rules; $accessing$ means the principal is executing granted privilege; $revokedAccess$ means the system denied the privileges of a principal based on runtime mutable attributes; and end means a principal terminates the access. Actions change the state of the system, which is the input alphabet. If the action is performed successfully the action is true, attributes of the principal, object and system are

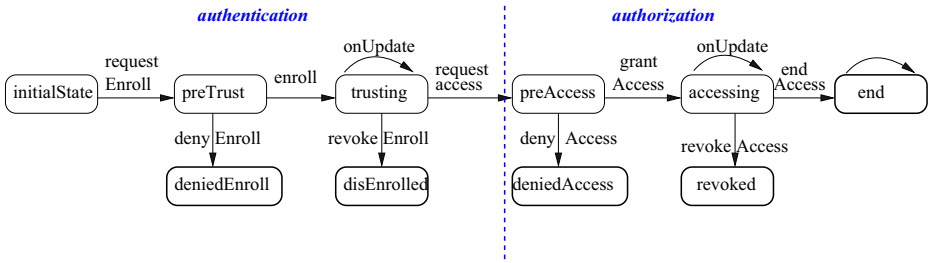


Figure 2. Trust-based UCON Model

assigned a new value. A series of actions are defined to change the status of a request. The transition from one state to another is triggered by an action, shown in Figure 2. $requestEnroll$ generates a new request when a principal tries to join the community. $denyEnroll$ rejects a request to enroll the community because the requester can not meet the minimum authentication or trust requirement. $enroll$ enrolls a principal to the community. $revokeEnroll$ revokes the allowed enrollment. $requestAccess$ generates a new access request. $denyAccess$ rejects an access request. $grantAccess$: grants an access request. $revokeAccess$ revokes an on-going and granted access request. $endAccess$: terminates an access request. $onUpdate$: updates the access request when mutable attributes or uncertain behaviors of a principal change.

4 Architecture of Trust-based Usage Control in P2P Delivery of Multimedia

When a principal p requests to execute a right r on an object o, attribute of the principal, permission (right r and object o, and an optional list credentials are submitted to *Secure Context Handler (SCH)* Module. The credentials may include signed trust-assertions (recommendations) from other users or a certificate signed by certificate authority. The *SCH* looks up the relevant contexts for the requested action, and queries the *Trust Calculator (TC)* component for a trust-value about principal p. Trust calculator calculates the trust value for requester based on both observed history and records in recommendation databases. A trust value is passed to *Access Control Manager (ACM)* Module for decision. The *ACM* looks up Access Control policies that entail several access control constraints. The *Constraint Service (CS)* Module and *Dynamic Manager (DAM)* Module evaluates access control constraints, e.g. time, location, memberships. Two categories of trigger events are possible to result in recalculation of trust value and reevaluation of access control policies. Recalculation and reevaluation may cause the revocation of current enrollment or on-going access. The *Evidence Handler (EH)* Module is listening to the peer reports about the misbehaviors of a requester. The negative report can include ignorance of obligation, dishonest behaviors, or the revocation of a requester's certificate. When the trust value of the request drops below a minimum threshold the on-going granted request will be revoked. The result of trigger event is notified to *SCH* and execution of request is cancelled. The *DAM* Module is listening to the attribute mutability of the principal, objects or a context after the permission is granted. For example, *DAM* Module can be triggered by certain events, for example, the subject left the group that entails the right. Once the *DAM* Module receives an event, the corresponding access control polices are re-checked by *ACM* if necessary (e.g., to allow an ongoing usage to continue or revoke it).

The update in *Trust Calculator (TC)* or *Dynamic Attribute Manager (DAM)* may revoke the granted permission. The mobile peers may report the dishonest behavior of requester or revocation of requester's certificate, so the trust value of request is dropped below a scalar. This update will be notified to Mobile Secure Handler and cancel the execution of request. After the permission is granted, Dynamic attribute manager will be trigger by certain events, the mobile node is moving out of range and not allowed for certain permission (reportAccident). Once the DAM receives an event, the attribute values of the object and subject are retrieved and evaluated and corresponding policies are rechecked by ACM if necessary (e.g., to allow an ongoing usage to continue or revoke it.)

5 Prototype Simulation

The architecture outlined in Section 4 provides the framework for the simulation program of usage-based access control model. This simulation works under the premise of several users who may request access to files owned by other users. Each of these users maintain modules included in trust-based usage control architecture, shown in Fig. 3. For every request, a trust value is calculated given past history and current recommendations for the requesting principal. Another factor in consideration is a risk assessment of the requested action assigned to each available file. Each owner assesses the risks based on sensitivity of his/her

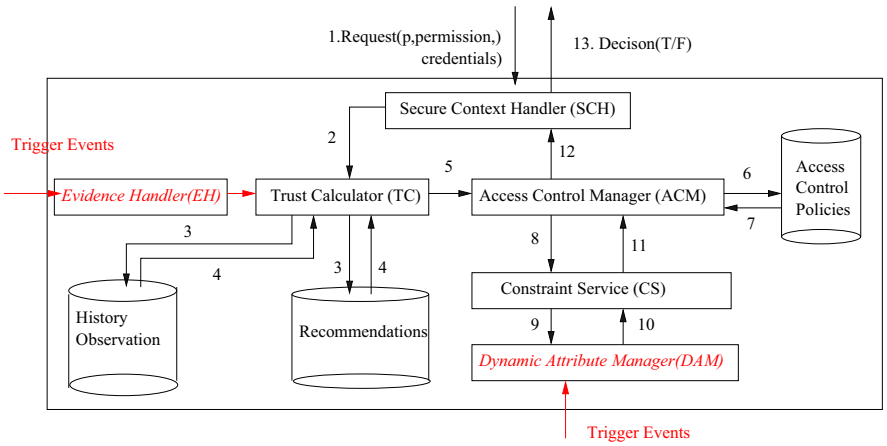


Figure 3. Trust-based Usage Control Architecture in P2P Delivery of Multimedia

file. Access to the file may be granted or denied based on trust evaluation, risk assessment and access control rules. If granted, the continuing usage of this access is contingent on maintaining the trust and risk values within the specified range. The on-going access may be revoked if the trust value is decreased below minimum threshold or access control rules are violated.

Besides successful request, several test scenarios are designed to test ability of our simulation program including how to evaluate trust, evaluate requests against access control rule, and react to evidence alerts and change of mutable attributes. First, a request fails the authentication when trust value of the request is lower than the minimum trust requirements. Second, a request passes the authentication but fails the authorization when a request does not meet the access control rules although its trust value is higher than the minimum trust requirements. Third, a request passes both the authentication and authorization; however, the on-going authorized request is revoked by negative evidence reports. Forth, a request passes both authentication and authorization; however, the on-going authorized request is revoked by mutable attributes such as change of domain or membership, which is triggered by events received from Dynamic Attribute Manager (DAM).

6 Discussion

Most recent research on access control include task-based authorization controls [17], team-based access control [9], role-based access control [8], temporal role-based access control [3], X-GTRBAC [5]. Recently, UCON [14] handles the mutability attributes of a principal or an object when the system makes decision for a request. All of them assume that a principal or an object is defined and represented by its attributes. This means that the identity, role or group of the subject can be identified through certain authentication mechanisms and that information about behaviors of a principal is certain. However, in a pervasive and collaborative environment, identity may not be identified. Moreover, identity itself can not convey priori information about the likely behavior of a principal. Behaviors of a principal may change between friendly and malicious when privileges are executed. A principal can not make access control decision only based on identity information because identity itself can not ensure friendly behaviors.

Reasoning and building trust for each peer allow peers to make decision when they are interacting with others in a peer-to-peer fashion. N. Li et al. [12] and W. Yao [21] use explicit incremental negotiation to establish mutual trust. An overview of trust management is discussed in [10]. Trust management has many applications in e-commerce areas such as works from Y. Atif [1] and P. Resnick et al. [15]. L. Xiong et al. [19] handles trust evaluation, especially the community-related context factors and transaction context factor of e-commerce. C. Zouridaki et al. [22] and L. Yang et al. [20] apply trust evaluation into routing protocols of mobile wireless ad hoc networks (MANETs).

R. Sandhu et al. [16] applies peer-to-peer access control to trusted computing, enforcing trust and hardware encryption. SECURE [6] project proposed the seminal ideas to handle trust and secure collaboration in uncertain environment. Their work can tolerate partial information, overcome initial suspicion to allow secure collaboration to take place by reasoning about trust and risk. N. Dimmock et al. [7] incorporate notions of trust into rule inference process of OASIS [2] , a policy-driven access control system. Mutable attributes, obligations, context and revocation of the authorization were not handled.

Both attribute mutability and uncertain behaviors of a principal are needed to be considered in collaborative resources sharing. In this work we integrate trust management into usage-based access control, which allows collaboration

when attributes of a principal are mutable or information on a principal's behaviors is incomplete.

7 Conclusion

We have proposed a framework to integrate trust management into usage-based access control. Our framework is designed to solve uncertainty and attributes mutability in a pervasive and collaborative environment. Our framework was simulated in the application of multimedia delivery in order to demonstrate the feasibility. The authentication and authorization to an on-doing request is checked constantly during the request. The granted request will be terminated if the trust value is lowered down due to negative peer reports or access control rules are not met due to attributes mutability.

References

[1] Y. Atif. Building trust in E-commerce. *IEEE Internet Computing*, 6(1):18–24, 2002.

[2] J. Bacon, K. Moody, and W. Yao. A model of OASIS role-based access control and its support for active security. *ACM Transaction Information System Security*, 5(4):492–540, 2002.

[3] E. Bertino, P. A. Bonatti, and E. Ferrari. TRBAC: a temporal role-based access control model. *ACM Transactions on Information and System Security*, 4(3), August 2001.

[4] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 41–52, New York, NY, USA, 2001. ACM Press.

[5] R. Bhatti, A. Ghafoor, E. Bertino, and J. B. D. Joshi. X-GTRBAC: an XML-based policy specification framework and architecture for enterprise-wide access control. *ACM Transaction Information System Security*, 8(2):187–227, 2005.

[6] V. Cahill, E. Gray, J.-M. Seigneur, C. D. Jensen, Y. Chen, B. Shand, N. Dimmock, A. Twigg, J. Bacon, C. English, W. Wagealla, S. Terzis, P. Nixon, G. di Marzo Serugendo, C. Bryce, M. Carbone, K. Krukow, and M. Nielsen. Using trust for secure collaboration in uncertain environments. *IEEE Pervasive Computing*, 02(3):52–61, 2003.

[7] N. Dimmock, A. Belokosztolszki, D. Eyers, J. Bacon, and K. Moody. Using trust and risk in role-based access control policies. In *Proceedings of the ninth ACM symposium on Access control models and technologies (SACMAT)*, pages 156–162, New York, NY, USA, 2004. ACM Press.

[8] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transaction Information System Security*, 4(3):224–274, 2001.

[9] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas. Flexible team-based access control using contexts. In *Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 21–27. ACM Press, 2001.

[10] T. Grandison and M. Sloman. Sloman: A survey of trust in internet applications. *IEEE Communications Surveys and Tutorials*, 3(4), 2000.

[11] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transaction Database System*, 26(2):214–260, 2001.

[12] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 114, Washington, DC, USA, 2002. IEEE Computer Society.

[13] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, New York, 1991.

[14] J. Park and R. Sandhu. The UCON usage control model. *ACM Transaction Information System Security*, 7(1):128–174, 2004.

[15] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation systems. *Commun. ACM*, 43(12):45–48, 2000.

[16] R. Sandhu and X. Zhang. Peer-to-peer access control architecture using trusted computing technology. In *Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 147–158, New York, NY, USA, 2005. ACM Press.

[17] R. K. Thomas and R. S. Sandhu. Task-based authorization controls (tbac): A family of models for active and enterprise-oriented autorization management. In *Proceedings of the IFIP TC11 WG11.3 Eleventh International Conference on Database Securty XI*, pages 166–181. Chapman & Hall, Ltd., 1998.

[18] C. e. a. Wu. rstream: resilient peer-to-peer streaming with rateless codes. In *Proceedings of ACM Multimedia Conference*, 2005.

[19] L. Xiong and L. Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transaction Knowledge Data Engineering*, 16(7):843–857, 2004.

[20] L. Yang, J. M. Kizza, A. Cemerlic, and F. Liu. Fine-grained reputation-based routing in wireless ad hoc networks. In *Proceedings of IEEE Intelligence and Security Informatics Conference*. IEEE Computer Society Press, May 2007.

[21] W. T.-M. Yao. Fidelis: A policy-driven trust management framework. In *Proceedings of the first International Conference on Trust Management*. LNCS, 2003.

[22] C. Zouridaki, B. L. Mark, M. Hejmo, and R. K. Thomas. A quantitative trust establishment framework for reliable data packet delivery in manets. In *SASN '05: Proceedings of the 3rd ACM workshop on Security of ad hoc and sensor networks*, pages 1–10, New York, NY, USA, 2005. ACM Press.

Flow Balancing Model for Air Traffic Flow Management

Bueno Borges de Souza[1]
Li Weigang[2]
Department of Computer Science - CIC, University of Brasília - Brazil.
buenobs@unb.br[1], weigang@unb.br[2]

Antônio Márcio Ferreira Crespo
First Integrated Center of Air Defense and Airspace Control - CINDACTA I, Brasília, Brazil
amf.crespo@uol.com.br

Victor Rafael Rezende Celestino
Brazilian Civil Aviation National Agency - ANAC, Brasília, Brazil
victor.celestino@anac.gov.br

Abstract

This paper describes a methodology based on Graph Theory and Artificial Intelligence (AI) for Air Traffic Flow Management (AFTM) problem. Flow Balancing Module (FBM) is proposed to manage air traffic flow through heuristics and dynamic adaptation. This model is integrated with a Distributed Decision Support System Applied to Tactical Air Traffic Flow Management (SISCONFLUX). The flow maximization technique is well known in Graph Theory and is adapted to an analysis model to determine which restrictive actions of flow control should be applied, and to what extent, in order to manage air traffic flow. The objective is to prevent or reduce congestions in diverse sectors within the airspace. With the scenario forecast and decision support modules, FBM supports the regulation of traffic flow to support controllers and other units within the SISCONFLUX.

1. Introduction

The First Integrated Center of Aerial Defense and Airspace Control - CINDACTA I in Brasilia controls about 50% of air traffic flow related to regular flights in Brazil [2]. The Flight Information Region in Brasilia (FIR-BS) is divided in twelve control sectors together with the Aerial Control Center of Brasilia (ACC-BS) to monitor and control the airspace over three regions: Brasilia, Rio de Janeiro and Sao Paulo. Every region with some sectors is managed by a supervisor, with obligation to perform decisions at occasions and each sector is monitored by a controller and an assistant [10].

The traffic flow management decisions taken by controllers and supervisors are based on their experience. The restrictive actions applied are made after an empirical analysis, not supported, therefore, with the help from any type of computational decision tool. Another important matter related to this issue is the impossibility to perform a quantitative evaluation of the impact of actions adopted in one specific sector on the traffic flow of adjacent sectors. As a consequence, it does not have an adequate forecasting level concerning the effect of the adopted restrictive actions on air traffic flow demand within FIR-BS as a whole. Thus, the inadequate sizing of actions applied by the ACC-BS will certainly imply in problems of traffic flow all over Brazil.

CINDACTA I, through ACC-BS and of Approach Control Centers (APPs), makes use of a set of systems capable of carrying out adequate air movements control in its area of responsibility. However, it does not have a specific system directed toward the tactical management and the synchronization of air traffic flow. This becomes even more critical when degradation of the usual control tools occurs, or some other factor which may cause significant modifications in the expected traffic flow, such as meteorological conditions, aeronautical incidents and/or accidents, amongst others. All these different factors can result in saturation of control sectors, characterized by the simultaneous permanence of fourteen or more aircraft in a sector [4]. The saturation of a sector can be conditioned by various factors, exampled by: the dimensions of the sector, the geographic position and the schedule of the day.

This research has the objective to describe a Flow Balancing Model (FBM) as a subsystem to integrate with Dis-

tributed Decision Support System Applied to Tactical Air Traffic Flow Management (SISCONFLUX). This system is being developed to assist the controllers and supervisors to efficiently management air traffic flow and to make the suitable decisions. FBM presents an analysis model using Graph Theory and Artificial Intelligence (AI) methods, with adaptations that make possible the discretization and the solution to this problem in each control sector.

The paper is organized in the following manner: section 2 presents a state of art of the research about AFTM. In section 3 the flow balance model using Graph Theory is described. Section 4 shows the architecture of the Flow Balancing Model (FBM) that integrates with SISCONFLUX. Finally, in the section 5 the final considerations of the research and the relevant references are presented.

2. AI Research in Air Traffic Flow Management - ATFM

ATFM is a task that involves the synchronization of air traffic flow in real time. The majority of the systems already considered by literatures [11, 12] present a centralized architecture, while [1] and [7] present solutions with a distributed character. All these solutions present excellent characteristics towards the efficient solving of the problem. However, they also study the problems related to the performance, that is, the necessary time for attainment of the solution and the formation of the "communication link".

The distributed method, however, presents huge management problems due to the exceeding number of message exchanges in situation of intense negotiation [1] and lack of physical structure – the Brazilian reality – that gives support to the implementation of negotiation techniques in distributed environments as the ones defined in [7].

In this context, the Ground Holding Problem - GHP emerges, which searches for the synchronization between adjacent sectors, by analyzing the set of landings and expected takeoffs inside the same area of supervision. This guarantees a better flow between these sectors. For this search, literature concerning the GHP suggests the use of integer linear programming [10] and [9]. This type of processing is computational expensive and becomes inefficient for application in real time.

Other works are suggesting methodologies based on dynamic programming for the attainment of better results [5], [8] and [14], the last two works suggest a representation of sectors and terminals, associated to the dynamic programming, in the format of graphs.

The latest proposals, suggest multi-agent architecture, implementation approach and software prototype of a multi-agent system for air traffic control within airport airspace capable of automatic detection of potential violations of safety policies by individual aircraft and corresponding incident management [6]; or Multiagent Simulation of Collaborative ATFM, where the authors evaluated several simple strategies for the Airline Operations Center (AOC) agents to select routes, using two different approaches, the Airline Planning approach and the Mixed approach [13].

3. Balancing Methodology

In this model a graph is defined as $G = (V, E)$ starting from a group of sectors that composes FIR-BS [14]. In this graph a multi-flow corresponding to the combination of the directional flows exists with origin and destinies associated to the terminals. Differently from the model proposed by Zhang [14], edges correspond to sectors and a path in the graph corresponds to a possible route. Each edge has an associated capacity and the vertexes represent the transition point between sectors. To illustrate the representation considers the Figure 1 that shows a partial cutting of FIR-BS with only three terminals. The set of paths between sectors

Figure 1. Part of FIR-BS

of the Figure 1 composes a multi-flow showed in the Figure 2.

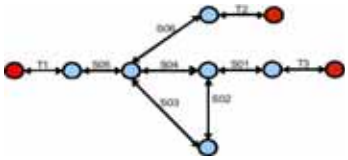


Figure 2. Multi-flow regard to a part of FIR-BS

The routes connecting among T1, T2 and T3 are:

$$\text{Routes}: \begin{cases} T1\ S05\ S06\ T2 \\ T1\ S05\ S04\ S01\ T3 \\ T2\ S06\ S05\ T1 \\ T2\ S06\ S03\ S02\ S01\ T3 \\ T3\ S01\ S04\ S05\ T1 \\ T3\ S01\ S02\ S03\ S06\ T2 \end{cases}$$

In a simplified way, one can identify three flows in the Figure 1: from $T1$ to $T2$ and $T3$, from $T2$ to $T1$ and $T3$ and

from $T3$ to $T2$ and $T1$. Suppose that f is the number of flows combined in the multi-flow, in the example $f = 3$. It is possible to build, from those graphs, an equivalent graph [3] that combines all these flows into a multi-flow, associating a source node to a destiny node (see Figure 3).

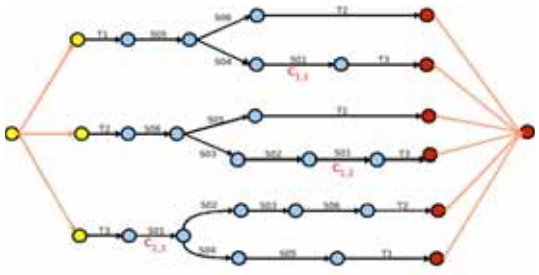


Figure 3. Joining of the flows generating the complete graph

Considering $C_{i,j}$ the capacity of the Sector i in the Flow j, limited by a constant value M in accordance with the legislation, and supposing a forecast of sector occupation of i of U_i, the balance of residual workload equals $L_i = M - U_i$. Under the following condition:

$$\sum_{j=1}^{f} C_{i,j} \leq L_i \text{ for all sector i} \tag{1}$$

where f is the number of flows associated to the multi-flow.

It is wanted that the flow be the largest possible, so solutions can be found making:

$$\sum_{j=1}^{f} C_{i,j} = L_i \tag{2}$$

Considering $k_{i,j}$ the fraction of flow j associated to the sector i the equation can be written as:

$$\sum_{j=1}^{f} k_{i,j} L_i = L_i \text{ for all sector } i. \tag{3}$$

that results in

$$\sum_{j=1}^{f} k_{i,j} = 1 \text{ for all sector } i \text{ and flow } f. \tag{4}$$

and is always truth by the definition of $k_{i,j}$. The problem is, then, to determine $k_{i,j}$ appropriately so that the multi-flow has a balanced distribution.

The distribution of $k_{i,j}$ is obtained by two manners: (1) Supervisors define flow quotas manually with the help of statistical data of stored distributions, previously adopted. The system will capture the controllers' experience and will keep that information for subsequent use. (2) Automatically by querrying previous accomplished actions, seeking the most suitable action for the current situation. The result retrieved supplies guidelines for the adjustment of $k_{i,j}$ and is submitted to the Evaluation and Decision Support Module (EDSM). The maximization of the internal flow f implies, in this case, the decrease of another flow due to the fact that the sum 4 is equal to 1. In that way the maximization of the multi-flow doesn't imply the maximization of all their components, and nor the opposite. Often there is interest in prioritizing certain flow in order to relieve an airport, for instance, the control can adjust $k_{i,j}$ to favor the flow of aircraft leaving terminal $T1$ in detriment of terminals $T2$ and $T3$. Based on graphs modeling technique, one can determine the maximum flow in the combined graph of flows composing the multi-flow. The flow measure is based on the *minimum cute*, that corresponds to maximum flow. The sum of flows in the combined graph (see Figure 3) equals the flow in the multi-flow [3]. The algorithm *Edmonds-Karp* is used for flow adjustments due to its simplicity and relative efficiency. The model considers the maximum number of 12 Sectors obtaining good performance for an algorithm complexity of $O(V \times E^2)$ [3]. The process will spend more time in adjustments of the time analysis showed in the next section. The flow adjustment algorithm does not consider the variation of the flow through time [3].

The model described in this article establishes a heuristic that redistributes the capacities based in sectors mean occupation time. The present problem with time analysis is that, in the context of air traffic, is associated with a discretization of the material that flows in the graph along time. In this case, a bad use of the sectors can happen, once it exists a time delay between the departure and the effective occupation of the sector. When reducing the departure frequency to solve the problem of saturation of a sector i which will be saturated in 20 minutes, for instance, some of the intermediate sectors $i-1$, $i-2$,..., can work with a departure frequency lower than it could be admitted. This becomes even worse when these sectors are in the intersection of several paths. Besides, there is also the need for adjustment of flow after departures, in other words, the saturation will happen with aircraft that already took off. In this case, flow restriction actions become critic. It is necessary to control the internal flow in transitions among sectors. With that purpose the Flow Balancing Module (FBM) makes a time analysis using queuing techniques and heuristics for each sector.

If the mean time to transpose sectors is all equal, the algorithm would not need to consider time. In this circumstance, flow would be approximately continuous and the model space associated with scenario forecast would be enough for the ground holding adjustments. There are

variations in sectors sizes, variations in aircraft speeds and deviations that alter the time to transpose different sectors. A solution considering all variants, unfortunately, will relapse in an excessive processing time. Therefore, the balance is being estimated, as follows: (1) There is a mean time $m'_{s_i} \leq m_{s_i} \leq m''_{s_i}$ to transpose sectors with a high limit and a low limit for all routes. (2) The time that the aircraft is in the sector b_{v_j,s_i} is known and accessible (v_j is the flight j and s_i the sector i); (3) Each flight v_j has its associated route known. (4) The *exit time* or the time that the aircraft will take to leave the sector a_{v_j,s_i} can be calculated with a good estimation by $a_{v_j,s_i} = m_{s_i} - b_{v_j,s_i}$; (5) The time to enter in a sector i is equals to the time to exit the sector $i-1$, being the sectors in the same route. It is taken, for each section, the orderly list of flights (aircraft) in the growing order of the *time to exit* the sector $Ls_i = a_{v_j,s_i} \geq a_{v_{j-1},s_i} \geq a_{v_{j-2},s_i}$.... The comparison of the time to exit the first element of the list Ls_i with the time to exit the first element in Ls_{i-1} is done. If the time to exit i goes below or equal to the one to exit $i-1$, the aircraft had left the sector before the previous enter (or at the same time), so the capacity of i can be increased by an unit. Repeating the same analysis for other aircraft on the list, if the time to exit i becomes longer than the time to exit $i-1$, in this case the capacity doesn't change, as that aircraft will stay in the sector in the considered period of time. Soon afterwards the sectors $i-1$ and $i-2$ are processed to obtain the desired capacity of these sectors. Through the analysis of these lists, FBM will be capable of obtaining the necessary time to wait en route in case i is saturated and the time to exit $i-1$ goes very short. In this case it will be suggested to EDSM a wait en route (orbit) for one of the aircraft on the list Ls_{i-1}, accompanied of an estimate of wait time.

4. FBM: Flow Balancing Module

4.1. FBM in SISCONFLUX

FBM will look for the possibilities that implies the ideal condition for air traffic flow. Such condition is characterized by the maintenance of the largest possible fluidity, restrictions of capacity of control sectors being observed, and the adjustment of those capacities so that the fluidity starting from some point in the area can be prioritized. The choice of traffic jam or saturation parameter will be determined by supervisors, taking into account technical and operational factors to apply, when necessary, restrictive actions to air traffic flow. Once the deliberations are defined, FBM will submit balance adjustments suggestions to the Evaluation and Decision Support Module (EDSM). EDSM evaluates those suggestions, informs the operational team about recommended actions and performs the learning procedure, which will allow the system to store a group of previous decisions and to adapt to the environment. After the decisions are taken and submitted to EDSM, the module also stores the actual scenario forecast associated with the group of actions taken. The actions are applied to the real scenario and the Monitoring and Scenario Forecast Module (MSFM) rebuilds a new scenario, considering all new information. This new scenario is, again, the input to FBM for reprocessing the need for restrictive actions in case they are necessary. FBM is divided in sub-modules according to the Figure 4 (related details are described in 4.2). The development includes three main submodules and two auxiliaries. The two auxiliaries submodules have the role to make communication with other system modules transparent to the main internal submodules. The three main modules accomplish indeed the processing of flow restrictions. Further on a detailed description of each submodule will be presented.

The development of FBM model for flow adjustment based on graph theory utilizes temporary adjustment techniques (see section 3). It is also associated with heuristics developed from actions commonly taken by the operators when performing dynamic adjustments of distribution of sectors capacities. In FBM, a knowledge base is built. It relates to the distribution of capacities on saturation scenarios, storing best flow distributions associated with restrictive measures used with more frequency for a given scenario. The mapping of the multi-flow in separate flows allows the adjustment so that certain flows are prioritized against others. Such prioritization will be the supervisors' responsibility, observing guidelines for better distributions already known. The more recommended flow restriction actions associated with each terminal are converted into frequencies of departures from specific origin points. This is justified by the fact that flight controllers work by limiting the interval among departures in a certain airport and not, specifically, with flight schedules. Specific adjustment actions of schedules are the aerodromes' responsibility, once the frequency of allowed departures is supplied. The recommended actions are not applied directly. Those actions are sent to the EDSM for analysis and submission to the supervisors who will appreciate suggestions, being able to accept them, and indeed apply them, or to request a new processing.

4.2. Submodules Description

The sub-modules in FBM are developed with the intention to distribute system inherent tasks, resulting in a better structural organization, facilitating overall understanding. In this section, a brief description of the functionality of each sub-module of FBM is presented (see Figure 4). The subdivision of FBM in submodules is developed with the intention to distribute system inherent tasks, resulting in a better structural organization, facilitating overall under-

Figure 4. Architecture of Flow Balancing Module

standing.

SMRD: This submodule has the role of receiving and/or looking for and formatting data for processing. It will work as a place for temporary data storage in case destiny submodules are busy. SMCG: This submodule has the role of building the graph associated to the current situation of the sector in CINDACTA I, using valid routes and to distribute the capacities in agreement with the information supplied by MSFM. SMAF: This submodule has the role of computing the ideal flow using the methodologies described in 3, to obtain recommended restrictive actions. SMMC: This submodule has the role of analyzing time forecast of aircraft permanence in sectors adjacent to the sectors congested and to suggest adjustments to speeds. It has also the function to record the critical state for analysis and to identify states lacking preventive actions. SMER: This submodule has the role of validating results obtained previously using defined guidelines, formatting results for the modules of the system and to send them. In the solution of the flow balance, the literature presents several algorithms of polinomial complexity and alternatives representation forms for graphs [3].

5. Final considerations

The flow balance technique utilized in the model presented involves the application of well-known algorithms [3] associated with heuristic adjustments. It is important to mention that FBM is a part of an integrated solution which foresees the projection of scenarios accomplished by MSFM and the application of reinforcement learning techniques accomplished by EDSM. The solution seeks the objective to be adherent to the centralized infra-structure existing for air traffic management in Brazil. As the future studies, it is recommended the application of negotiation techniques based on Game Theory to solve internal conflicts in the analysis of the critical restrictive actions, where one can seek a better global balance in order to distribute the damage of the restrictive actions among sectors and not to punish one or another determined sector.

References

[1] D. P. Alves, L. Weigang, and B. B. Souza. *Reinforcement Learning to Support Meta-Level Control in Air Traffic Management*, volume 1, chapter Reinforcement Learning - Theory and Applications, pages 409–424. Vienna: ARS publishing, 2008.

[2] CGNA. O controle do espaço aéreo - principais atividades. Publicação, Departamento de Controle do Espaço Aéreo, Rio de Janeiro, 2005. Vários autores.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill Book Company, 1998.

[4] A. M. F. Crespo, C. V. d. Aquino, B. B. d. Souza, L. Weigang, A. C. M. A. d. Melo, and D. P. Alves. *Sistema Distribuído de Apoio A Decisão Aplicado ao Gerenciamento Tático do Fluxo de Tráfego: Caso CINDACTA I.* In *VI Simpósio de Transporte Aéreo (VI SITRAER 2007)*, volume 1, pages 317–327, 2007.

[5] P. Dell'Olmo and G. Lulli. A dynamic programming approach for the airport capacity allocation problem. *IMA Journal of Management Mathematics*, 14:235–249, 2003.

[6] V. Gorodetsky, O. Karsaev, V. Samoylov, and V. Skormin. Multi-agent technology for air traffic control and incident management in airport airspace. In *Proceedings of 5th Workshop on Agents In Traffic And Transportation, @ Autonomous Agents and Multiagent Systems*, Estoril, Portugal, 2008.

[7] M. Heymann, G. Meyer, and S. Resmerita. A framework for conflict resolution in air traffic management. In *Proceedings, 42nd IEEE Conference on Decision and Control*, volume 2, pages 2035–2040, Maui, Hawaii, December 2003.

[8] Z. Ma, D. Cui, and P. Cheng. Dynamic network flow model for short-term air traffic flow management. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans*, 34(3):351–358, May 2004.

[9] A. Mukherjee. *Dynamic Stochastic Optimization Models for Air Traffic Flow Management*. PhD thesis, University of California, Berkeley, 2004.

[10] J. A. Rizzi. Um modelo matemático de auxílio para o problema de controle do tráfego aéreo. Master's thesis, Instituto Tecnológico de Aeronáutica - ITA, São José dos Campos, SP, Brasil, Maio 2003.

[11] U. R. Schlatter. Real-time knowledge-based support for air traffic management. *IEEE Intelligent Systems*, 9(3):21–24, Junho 1994.

[12] L. Weigang, C. J. P. Alves, and N. Omar. An expert system for air traffic flow management. *Jornal of Advanced Transportation*, 31(3):343–361, 1997.

[13] S. R. Wolfe, P. A. Jarvis, Y. Enomoto, F, and M. Sierhuis. *Comparing Route Selection Strategies in Collaborative Traffic Flow Management.*, chapter in Intelligent Agent Technology (IAT 2007). IEEE press, Fremont, USA, 2007.

[14] Z. Zhang, W. Gao, and L. Wang. Short-term flow management based on dynamic flow programming network. *Journal of the Eastern Asia Society for Transportation Studies*, 6:640–647, 2005.

VisRFID: Visualizing Customer Behavior in Geotemporal Space using RFID Technology

Beomjin Kim, Keith Bock, Michael Burton, Rod Strong, Benjamin Aeschliman
Department of Computer Science
Indiana University-Purdue University
Fort Wayne, IN, U.S.A.

Abstract

Understanding customer activities in retail stores is important information for organizing merchandise and planning marketing. This paper presents a prototype system that helps analyze the shopper's movements in retail spaces. We have utilized RFID technology to collect spatial and temporal data linked with the users' behaviors without violating their privacy. Visualization techniques are applied to the geotemporal data to present a large amount of information intuitively and informatively. The graphical illustration of the customers' traffic allows us to understand movement trends and to identify both frequently visited (hot spot) and unpopular (cold spot) locations in a store. The resulting output will immediately contribute to designing store layouts and maximizing store performance. The proposed system that utilizes visualization techniques with RFID technology can effectively monitor human activities in a variety of information spaces.

1. Introduction

RFID (Radio Frequency IDentification) technology uses radio waves to wirelessly access information stored in a small portable electronic identification tag – either a passive RFID tag or an active RFID transponder [1]. A microchip in an RFID tag, which can be attached to virtually any kind of object, contains a unique serial number and additional information about the object. An RFID reader generates a radio-frequency signal and transmits it to surrounding tags. Nearby passive RFID tags receive and modulate the signal, and then backscatter radio waves to the reader. Although this technology has been around since the second World War [2], the devices had previously been too expensive to be used in consumer applications – it is only in recent years that RFID technology has become usable to the public.

Today, RFID systems are used in a variety of applications to cut costs and increase efficiency in existing business environments. RFID technology provides benefits in a variety of situations such as inventory control, tracking items in warehouses, supply chain operations, and identification processes [1, 13]. It has also been used for implementing security systems, health care systems, and payment systems [14]. Wearable RFID readers and RFID transponders augmented with sensors allow us to detect objects and trace human motions [3]. The science museum in San Francisco used an RFID system to improve visitors' learning at the museum. Visitors put their RFID cards to readers attached to displays for taking pictures and acquiring information about exhibits they visit [4]. Visitors' log files are utilized to study their interests and preferences of presentations.

Although RFID has been actively used in many business applications, not much research has been conducted to graft RFID technology with information visualization. Shuping and Wright (2005) developed a GeoTime visualization that utilized RFID to track supply chains for the Department of Defense [5]. Their proposed system tracked the route of military supplies attached with RFID tags over location and time. The collected data was plotted in a three dimensional (3D) space where the XZ plane was used to display the path of supply shipments and the Y-axis was mapped to a temporal axis. Interconnected lines between data points in a geotemporal space produced an image showing where the shipments were and where they had been while in transit. The system provided functionality to visualize the data based on a time range, allowing users to verify that the shipments followed the intended path and were staying on schedule. Combined temporal and geospatial displays also made it possible to find any bottlenecks in the shipment route. The GeoTime visualization was mainly designed to show a single path in a 3D space which was projected onto a 2D screen. Thus, it was not an inherent method for drawing multiple paths without the introduction of overlapped lines. In addition, without any interactive capabilities, the projection of 3D data onto a 2D plane causes difficulty in understanding and interpreting the information.

Store layouts and the strategic placement of products are important factors for attracting target customers, optimizing store performance, and improving customer convenience [6]. Consumer researchers have studied various types of factors such as color, music, layout, signage, and fixtures to understand the influence each of these have on customers' reactions [7, 8]. However, previous studies commonly

collected data through questionnaires and thus, the reliability of the results heavily depends on the quality of responses [9].

Analyzing customer movements in retail stores, especially to identify poor performance areas, would be valuable information for modeling successful store layouts while considering the variability of customer activities. In order to obtain reliable data, Newman et al. (2002) used modern technology – closed-circuit television (CCTV) security cameras installed around the store – to acquire data associated with customer behaviors in retail spaces [6]. They applied image analysis techniques to recorded video footage to identify and count customer movements within the store. The developed application systematically traced customer traffic that would contribute to maximizing store performance and planning store layouts. Although this method anonymously tracked up to 20 customers at a time, it still not only experienced an ethical dilemma, but also required human intervention for editing, and thus had limitations on large scale data collection.

RFID technology can be an effective solution to monitoring customer movements in a retail store without worrying about privacy issues. The main goal of this project is the development of a prototype system to collect and visualize human activity in an efficient and scalable manner without affecting the privacy of consumers. The proposed system, called VisRFID, consists of three components responsible for collecting customers' locations over a period of time, translating it into a useful data set, and processing the data set in order to glean useful insights from it. The system utilizes emerging RFID technology to collect data associated with users' movements in retail spaces. We apply data visualization techniques to present a large amount of collected data in a spatiotemporal space intuitively through a visual abstraction. From the visually summarized data, consumer researchers can easily recognize consumers' traffic patterns associated with store arrangements. This will help to optimize customer traffic, identify hot and cold spots of the store, and improve space utilization. By expanding data collection, the compiled information can be used for a market analysis. Knowing the correlation of traffic patterns to purchased items is important to determine the design of stores, catalogs, and online shopping stores. The analyzed data will be valuable for planning marketing strategies while reflecting the preferences of target customers.

2. Method

The VisRFID system utilizes three major hardware components – RFID tags and readers, data transmitters, and a central server. The VisRFID consists of three major stages to visualize customer activity in a retail space. The data collection stage reads tags in the retail store. The data archiving stage transmits tag IDs to the server wirelessly for storage in a database. Finally, the visualization stage transforms collected spatiotemporal data to a graphical illustration.

2.1. Data Collection and Archiving (Stages 1 & 2)

We positioned passive RFID tags assigned with unique IDs in a space to be analyzed. In order to convert a tag ID to the associated coordinate in the space, each ID is registered and linked to a specific location in the server application. Additionally, each tag could be linked with products around the tag. The tag position and the distance between tags are factors to be decided depending on the strategic purpose of the application. A larger number of tags with a smaller sampling interval will provide higher quality data, but it will also increase the chance of tag collisions and generate data storage issues.

In our system, tags are positioned in a grid layout with the tag orientation parallel to the line of the customers' path. Tags along a straight path maintain equidistance. We allocated more tags to areas where the user can move in multiple directions. Shorter distances between RFID reader and tag improves the identification accuracy. To increase the proximity between the readers and tags, passive RFID tags are attached at both sides of the floor along shelves. The reader is mounted at the bottom of a shopping cart.

The collection of data was accomplished using an RFID reader connected to a PDA (Personal Digital Assistant). When a shopper moves along the aisle with a shopping cart, the attached reader collects the closest tag ID in accordance with the customer's location. At a predefined time interval, the PDA communicates with the RFID reader to retrieve the tag ID nearest to the reader's location. At each time interval, the PDA then sends the tag ID and the reader ID wirelessly to an edge server. When a customer stays at a certain spot over a long time period, the reader transmits the same tag ID multiple times. The server application converts the transmitted ID to an XZ coordinate using a lookup map. It then saves the position with a timestamp to a database for use in the visualization application.

2.2. Visualization (Stage 3)

Considering the large scale of collected data, analysis would be a very difficult task without proper abstraction or navigation tools. Information visualization is an effective solution for presenting a huge amount of data on a limited screen space. By presenting the data compactly and intuitively, it will not only make the evaluation easy, but also assist the analyzer's understanding of data by exploiting their visual perception [10].

We applied four different techniques to visualize acquired data. First, traditional line plotting on a 2D image was employed. The physical space is mapped to an I by J pixel image on the XZ plane. The positional data of a customer defines vertices on the 2D image and two successive vertices (X_i, Z_i) and (X_{i+1}, Z_{i+1}) at time t_i and t_{i+1} draws a line segment on the image. The linked line segments in time represent a user's movement in a retail store. Figure 1 shows an example view of a 2D image that plots customers' movements. The solid areas in the figure represent the fixtures in a retail store.

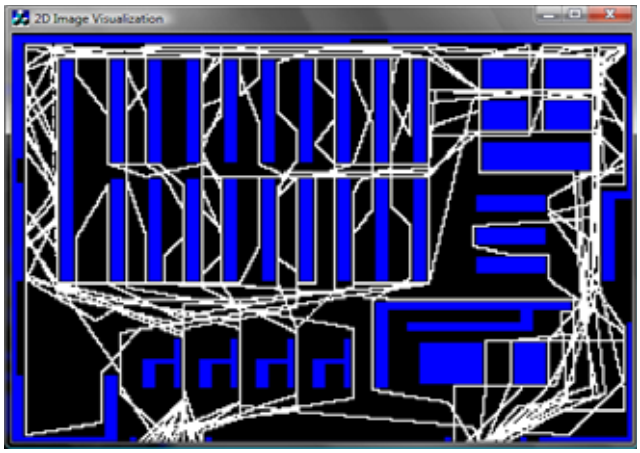


Figure 1. Plotting users' activities in a 2D image.

The second approach utilizes the GeoTime-type visualization [5, 11]. It adds a third dimension to the 2D image to show an additional attribute (time). The retail space is mapped to the 2D image on the XZ plane and the Y-axis represents the time. In this 3D plotting technique, a shopper's location at a certain time defines a point in a spatiotemporal space. Two successive vertices in time construct a line segment. The concatenated line segments in sequence correspond to a customer's passage in a retail space over a time. The perspective projection of the data in 3D space shows a shopper's movement on a 2D projection plane. Figure 2 is an illustration that displays multiple users' activities within a geotemporal space onto an image. With the installed interactive tools, the user can rotate and magnify a portion of the image for in-depth analysis.

Although the introduction of the third dimension addresses extreme data overlapping in 2D plotting, the GeoTime visualization is still too complicated for analyzing multiple users' movement trends from a single illustration [15]. In addition, considerable user interaction with the 3D space is required for seeing the area farther from the viewer, which is blocked by the data closest to the viewer. To overcome this, we applied mural-type visualization to the GeoTime image to improve the visibility of the overlapped area. An Information Mural is a visualization technique to create a view of a large data set that fits within a limited space, attempting to mimic the full data visualization without an excessive loss of information [12]. Mural-type visualizations are founded on a computer graphics technique, antialiasing, which computes the shade of pixel based on the weighted area sampling with overlapping weighting functions [17]. Instead of applying overlapping weighting functions that cause image blurring, we used a nonoverlapping weighting approach which will produce a sharper output and also decrease the computation time for constructing the mural style visualization. A data space is divided into M x N subspaces that are associated with M x N bins. The data points belonging to a specific subspace are mapped to the corresponding bin which is represented as a pixel on the visualized image. The number of data points in a bin relative to the bin having the highest number of data points defines the shade of the bin's condensed pixel (Figure 3). The visualized image makes it easy to interpret and identify both well-liked and unpopular areas.

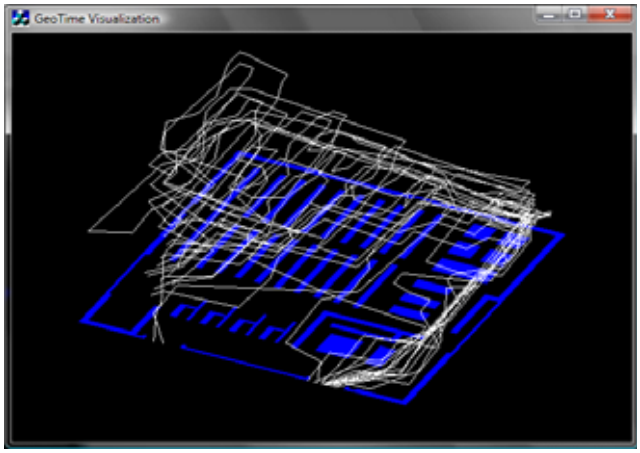


Figure 2. Plotting users' activities in a geotemporal space.

In order to apply this concept to the spatiotemporal data, the line segments representing users' movements in 3D space are projected on the XZ plane using an orthographic parallel projection. The XZ plane representing a retail area is divided into M x N subspaces which are mapped to M x N bins. Each projected location in a specific subspace of the XZ plane increases the frequency of the corresponding bin by one. Each bin represents a unit pixel on the m x n image to be visualized. A scaling procedure normalizes the frequency of each bin relative to the bin with the highest frequency assigning values between 0 and 1. The normalized value determines the intensity of an associated pixel in the visualized image where 0 represents the darkest available color and 1 represents the brightest intensity of the grayscale.

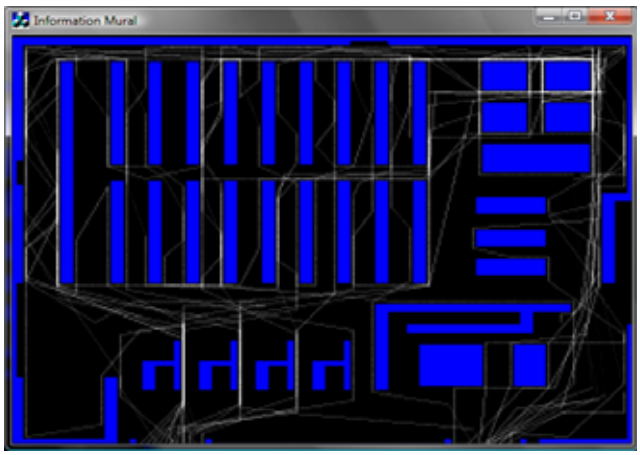


Figure 3. Plotting users' activities via mural-type visualization.

Figure 3 displays users' behaviors in a retail store over a time period using the method explained above. The solid boxes of the image represent the fixtures in a store. As shown in the figure, the popular areas in the store are represented with a brighter intensity. The user will spend more time at a position to look around or pick up items compared to when they are just moving to other locations. The total amount of elapsed time at a certain spot can be correlated with the shopper's interest of items around the area. When a customer stays at a certain spot for a while, the VisRFID system increases the intensity of that pixel relative to the amount of time elapsed. This will be important information for identifying and analyzing hot spots within the data set.

Displaying the customer's path chronologically will be valuable information for identifying an order to shoppers' habits. The VisRFID system employs color coding to show the user's movement chronologically. The total amount of shopping time of an individual customer is normalized. Two primary color components in RGB color scheme, red and green, are used to represent the color code for a specific time. The time T_0 when the user starts shopping has a color (G_{max}, R_{min}) where G_{max} is the brightest hardware green intensity and R_{min} is the darkest hardware red intensity. Meanwhile, the time T_n is mapped to a color (G_{min}, R_{max}) that represents the time when the user leaves the retail store. The linear color interpolation between the two sets of colors, (G_{max}, R_{min}) and (G_{min}, R_{max}), defines a color, $C_i(G_i, R_i)$, that represents the color at time T_i. Based on the mural technique described previously, the VisRFID system computes the intensity at a location j of the image, $Mural_j$. The system also computes a color for each shopper at j independently and determines a $Sum(G_j)$ and $Sum(R_j)$, where $Sum(G_j)$ represents the summation of all green intensities at j. $Sum(G_j)$ and $Sum(R_j)$ are compared and the ratio of the larger sum to the smaller sum is used to determine the final color. In order to minimize the influence of blue color, the mural intensity of blue is set to the smaller intensity value between the two colors of the location. For example, when $Mural_j = C$, $Sum(R_j) = \alpha$, $Sum(G_j) = \beta$, and $Sum(R_j) > Sum(G_j)$, then the mural color shading at location j is (C, $C * (\beta / \alpha)$, $C * (\beta / \alpha)$) in the order of red, green, and blue. The result of this algorithm will be a visualization with green shading where most shoppers begin their movement and red shading where most users finish.

Figure 4 shows users' chronologic path movements in a retail space where the majority of movements start at the lower right corner (green), moving in the counterclockwise direction, and exit toward the lower left corner of the image (red). By presenting customers' shopping routes based on normalized time span, it allows market analysts to compare users' chronological movement patterns and identify the commonalities in the timing of their movements. For instance, the people may tend to visit the freezer section later than the fruit section to avoid having their ice cream melt.

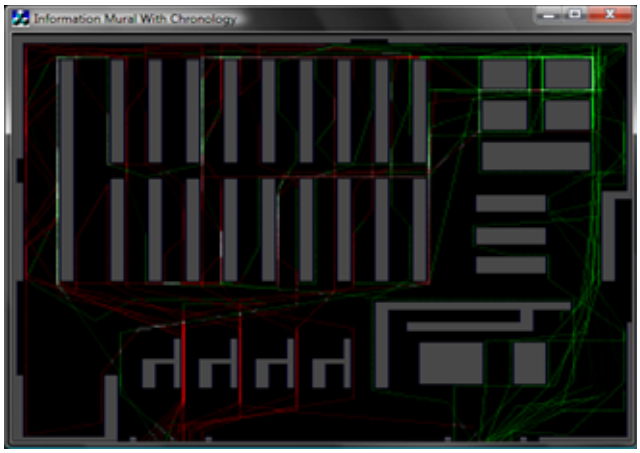


Figure 4. Plotting users' chronologic activities via mural color shading.

3. System Configuration

The VisRFID system was implemented with three major hardware components and custom software: RFID tags and a reader for data collection, a PDA for data transmission, and a PC server for data compilation and data visualization. EPC Class 1 Gen 2 tags (ALL-9440-02) produced by ALIEN Technology have been used to track customer positions in the working space. Gen 2 tags respond to an ultra-high frequency signal that has longer travel distance, but tends to be more directed, requiring a clear path between the tag and the reader. The ultra-high frequency tags can be activated from 10 to 20 feet away from the reader.

To reduce interference and maintain constant distance between tags and reader, RFID tags were attached on the floor and the RFID reader was mounted on a mini-cart and positioned to emit the RF signal toward the floor. We used the SkyeModule™ M8 reader that uses an RF range of 860-960 MHz to read passive tags. In our lab test, the reader could activate tags within the range of four feet, which is long enough to be applied in retail space. The reader is small enough to be operated with a 9V power supply.

The reader was connected to a PDA, a Dell Axim™ X51 equipped with wireless connectivity, via an RS-232 serial cable. At a predefined time interval, the PDA would retrieve a tag ID from the RFID reader. The PDA then sent the collected tag ID and the reader ID wirelessly to an edge server. The edge server converted the tag ID to the corresponding coordinate in the retail space and stored it with a timestamp to a database.

Custom software for the visualization application and the data transmission between the PDA and edge server was implemented in Visual C++ 2005, using MFC and OpenGL. For the pilot study, we constructed a testing environment that simulated the layout of a retail store in the lab. An array of RFID tags were attached on the floor of the lab and the reader and PDA were mounted on a wheeled cart to allow for easy movement around the tag array.

4. Results

Four different outputs were generated to show users' activities in a spatiotemporal domain through graphical illustrations. The outline of a sample store along with any large fixtures such as shelves and cashier checkout counters was also displayed on all four images. The first image shows customers' movements in a 2D plane where each shopper's movement in the retail store is plotted with connected line segments in time sequence (Figure 1). As expected, because of the complexity and overlapped plots, it is difficult to understand or interpret the traffic flow in the store. The second image plots the customers' behaviors in 3D space where the XZ plane represents the store coordinate and the Y-axis represents time (Figure 2). This GeoTime visualization somewhat untangles overlapping movements, but activities nearer to the viewer still block the movements behind those activities. These obstructions hinder an analyst in finding hot spots without manipulating the viewing orientation. The third image displays the same information plotted in Figure 1 and Figure 2 based on the mural type visualization. By resolving overlapped paths without losing the information magnitude, the mural style shading on the 2D map allows us to recognize hot spots within the data set intuitively. Finally, by integrating customers' movements in a time sequence with the mural-type image, the resulting output not only presents users' activities in a geotemporal space but also shows their chronological movements through the use of color shading (Figure 4).

5. Discussion and Future Research

The proposed system introduces an approach to utilize RFID technology for analyzing customers' movements in geotemporal space. Unlike existing approaches, RFID technology makes it possible to track users' behaviors in a retail space effectively without worrying about one of the main issues in the existing approaches – the violation of customers' privacy [6]. Although the developed application works well in a lab environment, several areas need further study before deploying the system in a practical environment.

It is of the utmost importance that we maintain the customer's privacy while monitoring their movements. In order to collect the shopper's behavior in a retail space anonymously, their identification must be separated from the data associated with their activities. For example, a user will randomly select a shopping cart and the store usage data should be processed independently from any customer loyalty card or credit card used during checkout.

Since the RFID reader travels with the customer, a hardware enhancement of a battery pack for operating the RFID reader is an essential task for monitoring their movements. The RFID reader powered by a battery pack should operate for a reasonable time period before needing a recharge. The RF signal propagation range has a close relationship to the amount of consumed power. In a retail store, to increase the proximity between the reader and tags, RFID tags can be attached either at the bottom of the shelves or at both sides of the floor along the shelves. The reader will be mounted on the bottom of the shopping cart to keep it from becoming inconvenient to the customer in their shopping activities. Smaller distances between the RFID reader and tags will both decrease the power consumption and improve the reading accuracy.

RFID tags and readers could alternately be deployed in the opposite configuration. Namely, readers would be positioned at the shelves and RFID tags attached to the shopping carts. Both configurations have problems to be addressed. The configuration mounting a reader to a cart may introduce two concerns. First, to provide mobility to the reader, it should be operated by battery power. Second, in a practical environment, the reader and PDA on the cart can get stolen or damaged by the weather when the cart is outside of the store on a rainy day. Meanwhile, the opposite design will introduce other significant issues such as greatly increasing installation costs and introducing inflexible scalability. Regarding the cost concern, the total number of RFID readers installed in a retail space may be positively correlated with the quality of data showing customer's behavior, potentially making it worth the greater investment. However, unlike repositioning tags attached on the floor, reorganizing readers fixed to shelves will be a much more challenging task, and non-uniform distances between the reader and moving tags on the cart will decrease the reading accuracy.

Another hardware issue to address before deploying the pilot system in practice will be handling signal collisions. RFID reading devices can experience two types of collision – reader collisions and tag collisions [16]. The readable distance of an RFID reader varies depending on the frequency range it uses. When there are multiple readers within an overlapped readable area, signals from readers in that region will interfere with each other. The advancement of RF technology, such as a reader retrieving tag IDs using a time division multiple access technique, will lessen RF signal collisions.

The omni-directional characteristic of radio waves from the RFID reader may activate multiple tags within its reading range. This will cause RFID tag collisions in which a reader will receive several reflected signals [16]. The proper layout of passive tags could be a factor that decreases tag collisions. Positioning a sufficient number of tags to trace users' activities correctly while maintaining proper distance among them will reduce the chance of tag collisions. A software enhancement that identifies the tag closest to the customer's position after analyzing reflected signal strengths and previous movements could be another solution to correct the problem.

The potential environments for applying the proposed applications are quite wide-ranging. We can employ the system to understand visitors' movements and preferences at public places including shopping malls, galleries, and museums. It is also applicable with minimal modification to other situations for studying factors associated with human

behaviors such as the movements of factory workers, chefs, and players in team sports. This motion related data will be an important resource to optimize moving routes at work places and to develop strategies for team play.

We expect further enhancements to the visualization software. The mural type visualization effectively projects 3D spatiotemporal data onto a 2D image, but it doesn't show directional information of users. The addition of directional information to the graphical illustration will further indicate trends in movement within the geotemporal space. Introduction of complementary colors to the current grayscale color scheme will be a possible approach to present directional information with users' routes. The visualized image will allow the store manager to understand the directional tendency that people take when navigating the store. Additional graphical attributes such as thickness, orientation, pattern, and overlaid images can be integrated to the visualization to represent other factors – for example, number of customers, correlations among hot spots and possibly purchased items. This could provide further understanding about customers' behaviors associated with store layout.

6. Conclusion

This paper introduces an end-to-end prototype system for collecting, archiving, and visualizing data related with customers' activities in a retail store. The utilization of RFID technology for collecting data makes it possible to acquire data associated with users' movements while addressing the privacy issue existing in previously proposed methods. The developed visualization software clearly shows customers' activities in a spatiotemporal domain projected onto a 2D image while addressing the issue of overlapped paths on the output. The visualized image would allow consumer researchers to recognize various trends within the data, particularly to identify "hot spot" and "cold spot" locations of a retail space effectively. It will immediately contribute not only in planning store layouts but also in setting up marketing plans. Our pilot application developed in a lab environment combines RFID technology with visualization techniques. As RFID technology matures and develops, it seems that the potential applications for it are almost endless, and we expect that the system presented here can perhaps be used to aid in the effectiveness of RFID technology in our society.

7. References

[1] Want, R., "RFID: A Key to Automating Everything," *Scientific American*, pp. 56-65, Jan. 2004.

[2] Landt, J., "Shrouds of Time: The history of RFID," *Association for Automatic Identification and Mobility publication [Online]*, https://www.aimglobal.org/estore/ProductDetails.aspx?ProductID=529, 2001 [Accessed Nov. 26, 2007].

[3] Smith, J., Fishkin, K., Jiang, B., Mamishev, A., Philipose, M., Rea, A., Roy, S., and Sundara-Rajan, K., "RFID-based techniques for human-activity detection," *Communications of the ACM*, Vol. 48, pp. 39-44, 2005.

[4] Hsi, S., and Fait, H., "RFID enhances visitors' museum experience at the Exploratorium," *Communications of the ACM*, Vol. 48, pp. 60-65, 2005.

[5] Shuping, D. and Wright, W., "GeoTime Visualization of RFID Supply Chain Data," *RFID Journal [Online]*, http://www.rfidjournal.com/whitepapers/1/5, 2005 [Accessed Nov. 26, 2007].

[6] Newman, A. and Foxall, G., "In-store customer behaviour in the fashion sector: Some emerging methodological and theoretical directions," *International Journal of Retail & Distribution Management*, Vol. 31, No. 11, pp. 591-600, 2003.

[7] Newman, A, Yu, D., and Oulton, D., "New insights into retail space and format planning from customer tracking data," *Journal of Retailing and Consumer Services*, Vol. 9, pp. 252-258, 2002.

[8] Yalch, R. and Spangenberg, E., "The effects of music in a retail setting on real and perceived shopping times," *Journal of Business Research*, Vol. 49, No. 2, pp. 139-147, 2000.

[9] Martin, C., and Turley, L.W., "Malls and consumption motivation: an exploratory examination of older Generation Y consumers," *International Journal of Retail & Distribution Management*, Vol. 32, No. 10, pp. 464-475, 2004.

[10] Card, S.K., Mackinlay, J.D., and Shneiderman, B., *Readings in information visualization using vision to think*, Morgan Kaufmann Publishers, 1999.

[11] Kwan, M., "GIS Methods in Time-Geographic Research: Geocomputation and Geovisualization of Human Activity Patterns," *Geografiska Annaler B*, Vol. 86, No. 4, pp. 267-280, 2004.

[12] Jerding, D. and Stasko, J., "The Information Mural: A Technique for Displaying and Navigating Large Information Spaces," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4, No. 3, pp. 257-271, 1998.

[13] Songini, M., "Wal-Mart offers RFID update," *Computerworld [Online]*, www.computerworld.com/mobiletopics/mobile/story/0,10801,109418,00.html, 2006, [Accessed Nov. 30, 2007].

[14] Anshel, M., and Levitan, S., "Reducing medical errors using secure RFID technology," *ACM SIGCSE Bulletin*, Vol. 39, Issue 2, pp. 157-159, 2007.

[15] Kapler, T., Harper, R., and Wright, W., "Correlating Events with Tracked Movements in Time and Space: A GeoTime Case Study," *Proceedings of International Conference on Intelligence Analysis [Online]*, https://analysis.mitre.org/proceedings/Final_Papers_Files/195_Camera_Ready_Paper.pdf , 2005 [Accessed Nov. 30, 2007].

[16] Jain, S. and Das, S., "Collision avoidance in a dense RFID network," *Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization*, pp. 49-56, 2006.

[17] Foley, J., Van Dam, A., Feiner, S., and Hughes, J., *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Company, 1996.

Acknowledgements

This research was partially supported through IPFW Grant-in-Aid of Research by The Office of Research and External Support at Indiana University-Purdue University Fort Wayne.

Analyzing Manufacturing Process Knowledge Flows with KoFI

Oscar M. Rodríguez-Elias[1], Alberto L. Morán[2], Jaqueline I. Lavandera[3], Aurora Vizcaino[4]
[1]*UNISON-Mathematics Department, Hermosillo, Son., Mexico*
[2]*UABC-Facultad de Ciencias, Ensenada, B.C, Mexico*
[3]*FAMOSA-Ensenada, Ensenada, B.C., Mexico*
[4]*ALARCOS Research Group, Information Systems and Technologies Department, UCLM. Spain*
omrodriguez@ciencias.uson.mx; alberto_moran@uabc.mx; jilavmac@efemsa.com; aurora.vizcino@uclm.es

Abstract

This paper presents the use of the Knowledge Flow Identification (KoFI) methodology as a means to improve a manufacturing process knowledge flow. KoFI was initially developed to analyze software processes. In this paper we illustrate how it can also be used in a manufacturing domain. The results of the application of KoFI are also presented, which include some lessons learned, and the design of a knowledge portal together with the results of an initial evaluation from the potential users of this portal.

1. Introduction

Knowledge is currently one of the most important organizational resources [2]. It is therefore important for organizations to search for ways to manage it. To accomplish this, knowledge management systems (KMSs) must facilitate knowledge workers with the knowledge they require from where it is created or stored, or capture and store knowledge to make it available for future use. It is necessary to understand how knowledge is flowing in the work processes, in this way it should be easier to identify the problems affecting that flow and, as a consequence, to propose possible solutions to improve the flow [5].

In this paper, we illustrate the manner in which the KnOwledge Flow Identification (KoFI) methodology [8] was used to analyze a manufacturing process, in order to improve its knowledge flow. The main reason for engaging in this study was to assist a manufacturing organization in two main aspects: 1) to improve the training of highly competitive personnel, and 2) to promote organizational learning. The main concern was to develop a KM system to assists the human resources training process, by making useful information and resources available to the employees to promote self-learning and knowledge diffusion.

In the accomplishment of the above goals, certain questions arose, such as: what knowledge is it important for the employees to have? Where does that knowledge reside? How can it be accessed? Which apects of such knowledge are being stored and where? Which are not being stored and why not? etc. To obtain initial answers to these questions and to propose a possible solution for the organization a study was carried out. In this study one of the organization's processes was analyzed using the KoFI methodology. The main results of this study are described here. The paper is organized as follows: Section Two summarizes the KoFI methodology. Section Three goes on to depict the analysis of the manufacturing process, while Section Four introduces a knowledge portal whose design was based on the results of such an analysis. Finally, Section Five presents the results of an initial evaluation of this portal, while Section Six concludes the paper.

2. The KoFI methodology

KoFI is a methodology focused on identifying and analyzing knowledge flows in work processes, following process engineering techniques [8]. It was defined to assist in three main areas: 1) to identify, structure, and classify the knowledge base of a studied process, 2) to identify the technological infrastructure that supports the process and which affects the knowledge flow, and 3) to identify requirements to improve the knowledge flow in the process.

In order to apply KoFI, it is necessary to define the specific process to be analyzed, and then model it. The process models are later analyzed following a four step process, as is shown in Figure 1. The process followed

is iterative, since each stage may provide information useful for the preceeding and successive stages. Thus, it is possible for the process model to evolve while it is being analyzed through KoFI. We shall now attempt to describe how each stage is carried out.

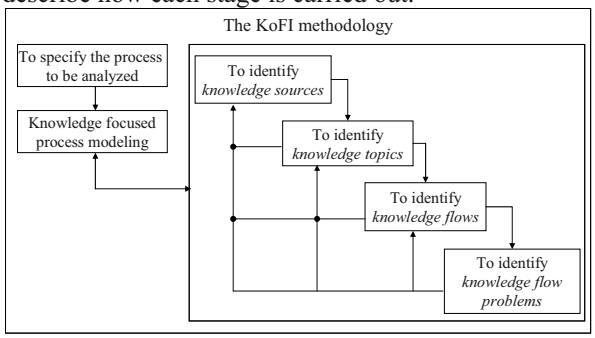


Figure 1: Stages of the KoFI methodology.

2.1. Knowledge focused process modeling

To model the knowledge involved in a process, it is convenient to use a Process Modeling Language (PML) which provides explicit representation of issues such as the knowledge consumed or generated in activities, the knowledge required by the roles participating in those activities, the sources of that knowledge, or knowledge dependencies [1]. In our study we used an adaptation of a highly used and flexible PML proposed in [7].

Since the focus of this paper is not on the modeling languages, we will limit ourselves to simply presenting the main activities carried out in KoFI.

2.2. Identification of knowledge sources and topics

These two steps focus on identifying the main documents and people involved in the process, that is, the main knowledge sources, and the knowledge that can be obtained form the, or the one required to accomplish the process' activities. We consider people as a knowledge source since they are the main source of tacit knowledge in a company. It is important that the identified sources and topics be organized and classified, for instance, by means of a taxonomy or an ontology in which the relationships between the elements of the process be represented. In fact, defining taxonomies is one of the first steps in the development of KMSs [6].

2.3. Identification of knowledge flows

In the third step we analyze how knowledge and sources are involved in the activities performed in the process. The main activities of the processes have, of course, been previously identified. Therefore, the process models help to analyze how knowledge flows through the process while the people involved perform their activities. Examples of this include knowing which sources are consulted, or which documents are generated while activities are performed. It is important to identify knowledge flows in activities and/or in sources. One example of this might be the transfer of knowledge from a person to a document.

2.4. Identification of knowledge flow problems

The knowledge flows identified in the previous stage are analyzed to discover problems which might be affecting them, such as whether the information generated from the activities is not captured, or whether there are sources that might help in the performance of certain activities, but which are not consulted by the people in charge of them. To do this, KoFI proposes the use of *problem scenarios*, which are stories describing the way in which a problem occurs [8]. These stories must particularly show how the detected problems affect the knowledge flow. Once the problem scenario is described, one or more alternative scenarios must be defined to illustrate possible solutions, and the manner in which those alternative solutions may improve the flow of knowledge.

3. Analysis of the manufacturing process

The KOFI methodology was used in a manufacturing process with the goal of detecting how this process could be improved from a knowledge management point of view. The study was conducted in a Mexican industrial company dedicated to the manufacturing of cans. We studied one of eight processes performed in one of nine departments in one unit of the company, specifically the process in charge of transforming the aluminum rolls into the first versions of the cans (known as the "Formation area"). Forty one people were involved in this process.

It is important to highlight that the company has documented all its processes, and follows standards for documenting almost all its activities. Moreover it has an ISO9001-2000 certification, so it was not necessary to develop detailed models of the processes. We simply focused on developing high level models to identify the main knowledge required for the central activities of the processes and to identify the main knowledge and information sources involved.

The data used to analyze the process was captured through interviews, and by analyzing documents and

information systems. Nineteen employees were interviewed by using the long interview technique, but adjusting the interviews to the following format: the general data of those interviewed, the main activities performed, and knowledge sources known by them, and their level of knowledge of the process. The duration of the interviews ranged, from 30 minutes to 2 hours, depending on the level of responsibility of those interviewed. A total of 119 documents and systems were also analyzed, of which 24 were discarded because they were duplicated.

3.1. Results of the analysis

The main results of the analysis of the process were classification schemas for knowledge sources and topics, which were later used as a basis for structuring a knowledge map from which a knowledge portal was developed. Additionally, the knowledge flow analysis phase helped us to identify the relationships between the various knowledge sources and topics, and the activities carried out in the process. These main results are next described.

3.1.1. Knowledge sources. The identified sources were very diverse, from process documentation to organizational norms. These were classified into: 1) documents, including three subcategories: process, technical, and organizational documentation; 2) information systems, including two subcategories: query, and transactional systems; 3) people, including four subcategories: staff, specialists, external clients, and internal clients; and 4) others, including two subcategories: problems analysis, and simulation tools.

3.1.2. Knowledge topics. The identified knowledge topics were also very diverse, ranging from organizational behavior to special machine maintenance. These topics were classified in three categories: 1) product line activities, including product quality, machine maintenance, operation, and information technology (IT) application; 2) organizational culture, including knowledge of the company; and 3) general knowledge, including resource management, IT management, personnel management, and other individual knowledge.

3.1.3. Knowledge flows. In this step we modeled the knowledge required in each activity of the process, the knowledge that each role required to perform these activities, and the knowledge sources consulted or generated in each activity, following an adaptation of the Rich Picture [4] technique proposed in [7]. These models helped us to identify the relationships between the knowledge sources and topics, and the activities of the process. This allowed us to create a knowledge map by defining the knowledge that might be obtained from each source, and by defining the activities in which the sources or the knowledge were being used or generated.

3.1.4. Knowledge flow problems. In the final step of KoFI, we identified two main areas of opportunity. First, it was observed that some areas of the process were not well supported with documentation. For instance, there was not enough documented information on the use of certain mechanical and electrical tools; therefore, that knowledge resided only in people's experience. An additional problem was the identification of important knowledge sources that were not being used; for instance, the company had some simulation tools that were not being used.

Based on the information obtained by applying the four steps of the analysis phase of KoFI, we decided to develop a knowledge portal which could facilitate access to the available knowledge sources , classifying them according to the activities in which they would be useful. It was also decided that the portal should provide access not only to documents, but also to other types of sources, such as information systems, or support tools, in order to promote the use of all the available types of knowledge sources of the company.

4. Designing the Knowledge Portal

From the analysis we created a knowledge meta-model which could be replicated to any area of the organization while achieving the same results.

4.1. Meta-model

The proposed meta-model, represented in Figure 2, comprises the knowledge types and sources involved in the knowledge generation and acquisition process.

In this meta-model the knowledge concepts are integrated with the knowledge topics and sources. The knowledge concepts are required, generated or modified by the activities within the study area, which are described as work definitions. In turn, these work definitions can be represented as processes, activities or decisions. Each knowledge concept/source association contains information about the knowledge level it requires. Finally, the available format and location for consulting each source are specified.

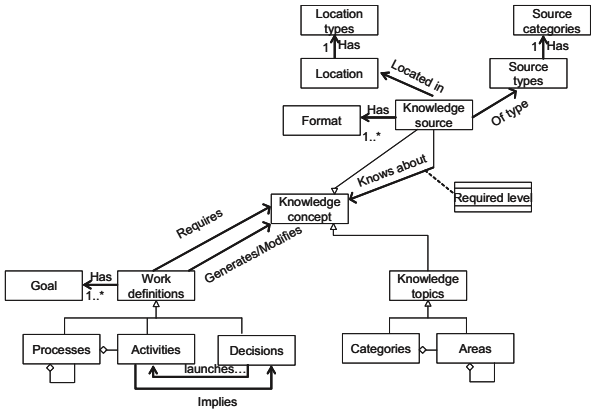


Figure 2: Meta-model of knowledge types and sources.

4.2 Knowledge portal structure

This meta model was used as a base to design the structure for a knowledge portal. Figure 3 shows the resulting general structure of the portal. This structure comprises a first level in which initial interfaces (pages) are accessible (e.g. home and registration pages). The second and third levels are pages which correspond to the manufacturing areas and sub-areas of the organization, respectively, according to the rich picture models developed during the analysis. The fourth level corresponds to pages on the processes that integrate each of the sub-areas identified from the involved knowledge flows. Finally, the fifth level presents all the identified knowledge sources for the specific process of the sub-area. This structure is representative of all and each of the manufacturing sub-areas, as identified during the analysis.

4.3 Knowledge portal UI design

The design of presentation and navigational features of the user interfaces (pages) of the knowledge portal also emerged from insights identified in the analysis and initial phases of design.

These include information about the identified knowledge flows, the main sub-areas of the organization, and the structure of the portal previously identified, which resulted in the options included in the menus and main layout sections of the pages. These allow users to find the required information by simply identifying the specific area in which information is generated or required, and following the resulting navigational structure (area → sub-area → process) to locate the specific knowledge source, instead of just alphabetically (or randomly) browsing through the information.

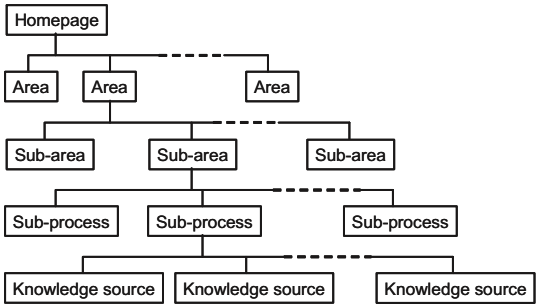


Figure 3: General structure of the Knowledge Portal.

Figure 4 depicts an example of the layout and content of a page from the current prototype for the "Formation" area.

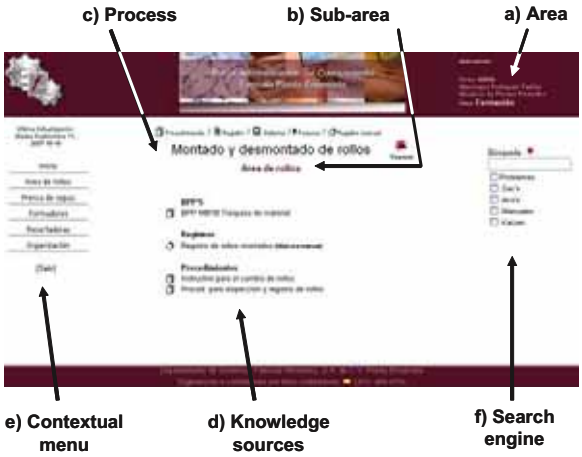


Figure 4: Example of the page contents and layout of the Knowledge Portal.

The information provided includes the name of the manufacturing area being consulted (4.a), the name of the specific sub-area (4.b), the name of the selected process within the sub-area (4.c), and most importantly, links to knowledge sources (and types) available for that process (4.d).

Additionally, the page includes a "contextual" sub-area menu to facilitate navigation through the information (4.e), which is always available while the user stays in that particular sub-area of the portal. Also, it includes a search engine (4.f) which allows a search to be performed by simply specifying a keyword on the required topic, and optionally, the "places" in which the information should be searched for.

The interface in Figure 4 represents the final destination for users looking for a particular knowledge source who, by following only three links

(area → sub-area → process), arrive at the knowledge sources (either documents, systems or people) required to perform their intended activities.

Finally, this design adheres to the organization's established standard guidelines for this kind of applications.

5. Preliminary Evaluation of the knowledge portal

We conducted a preliminary evaluation in one of the production areas to both determine the impact and acceptance level of the users on the system, and to provide support for the decision-making process concerned with the continuation of the system's implementation in other areas of the organization. The evaluation considered aspects concerning perception of usefulness and ease of use [11].

The evaluation consisted of 1) an introductory session, in which the system was presented to the users and its functionality was demonstrated to them. This included examples on how to search for and retrieve knowledge sources by means of navigating through areas, sub-areas and processes, as well as through the search engine; and 2) the application of a questionnaire containing 12 questions referring to perception of usefulness (6) and ease of use (6). Each evaluation session (induction and application of the questionnaire) was done in approximately one hour.

The subjects of the study were 41 employees of the "Formation" area for which the prototype was developed. The subjects included leader mechanics, process mechanics, operators and process engineers, whose participation was voluntary. The sample was divided into 4 groups according to the natural operative processes (3 groups of ten people and 1 of eleven). The application process of the evaluation was completed in three days.

5.1 Analysis and discussion of evaluation results

The subjects had positive appreciations with regard to the knowledge portal, as is reflected in their answers in the questionnaire. Table 1 shows the answers to the questions about the perception of usefulness of the tool. The users perceived that the portal would allow them to increase their productivity and to perform their tasks more easily (82.93% "Agree" in both cases), although some of them had doubts regarding the fact that this would increase their productivity (24.39% "Have Doubts"). Only one person (2.44%) "Disagreed" that the tool would help him/her to complete his/her tasks faster.

Table 1: Perception of Usefulness.

Question	Agree (%)	Have Doubts (%)	Disagree (%)
Complete the task faster	78.05	19.51	2.44
Increase task performance	82.93	17.07	0.0
Increase productivity	75.61	24.39	0.0
Improve efficiency	80.49	19.51	0.0
Ease the task	82.93	17.07	0.0
Is useful	87.80	12.20	0.0.

Table 2: Perception of Easy of Use.

Question	Agree (%)	Have Doubts (%)	Disagree (%)
Learning to browse	85.37	14.63	0.0
Finding information	60.98	39.02	0.0
Clear user interfaces	65.85	34.15	0.0
Flexible interaction	65.85	34.15	0.0
Becoming an expert	53.66	46.34	0.0
Is easy to use	68.29	31.71	0.0.

Table 2 shows the answers to the questions about the perception of ease of use. As can be seen, although most of the users perceived that it was easy to learn to browse through the information (85.37% "Agree"), some had doubts concerning the ease of finding information (39.02% "Have Doubts"), and even more users had doubts concerning becoming experts on the use of the tool (46.34% "Have Doubts"). A possible explanation could be that a little more than a third of the users had doubts concerning the clarity of the presented interfaces, as well as about the interaction flexibility that these provide (34.15% in both cases).

In general, most of the users considered the knowledge portal as a useful (87.80% "Agree" – Table 1) and easy to use tool (68.29% "Agree" – Table 2) for the accomplishment of their work.

6. Discussion and concluding remarks

Integrating KM into work processes is one of the main concerns in the KM community [9]. Several works related to the integration of KM into work processes can be found in the relevant literature (e.g. [1, 3, 10]). Most of the approaches we have found are either orientated towards developing specific KM

systems, or require special tools or PMLs for their use. Before proposing a specific approach for managing knowledge in an organization, it is important to analyze the organizations' work processes from a knowledge flow perspective [5], since supporting knowledge flow should be the main focus of KM. The main contribution of the present work is the use of an approach which takes this observation into consideration. We have illustrated how the KoFI methodology [8] may be useful for proposing means to improve the knowledge flow in a manufacturing company. This should be accomplished not only by developing new systems, changing organizational culture, and so on, but also by integrating the current infrastructure and the real work being done by the people in charge of the organizational processes.

The main result of the study was illustrating the usefulness of the KoFI methodology in a manufacturing setting; particularly for the design of a knowledge portal based on the real work structure of a company. The portal integrates the knowledge sources available, and presents them to the users by following an organizational structure which emerges from the application of the different steps proposed by KoFI. Even though more research is required to evaluate if the portal will allow the company to improve the training of highly competitive personnel, and to promote organizational learning, the preliminary evaluation of this portal has led us to believe that it could help to accomplish this, since such a portal was considered to be highly useful and used by the employees of the company. As future work, we are planning to apply the KoFI methodology to the analysis of all the company's other processes, in order to extend the use of the portal to the entire organization. This should help us to continue evaluating the benefits and limitations of KoFI.

Acknowledgements

This work is supported by UABC, project 0191 of the XI Convocatoria Interna de Proyectos, the MELISA project (grant PAC08-0142-3315) financed by the Consejería de Educación y Ciencias de la Junta de Comunidades de Castilla-La Mancha, and CALIPSO (TIN20005-24055-E) supported by the Ministerio de Educación y Ciencia (Spain). The authors acknowledge the support provided by FAMOSA-Ensenada, for the realization of this project.

References

[1] P. Bera, D. Nevo, and Y. Wand, "Unravelling Knowledge Requirements through Business Process Analysis," Communications of the Association for Information Systems, vol. 16, pp. 814-830, 2005.

[2] K. Ichijo and I. Nonaka, "Knowledge Creation and Management: New Challenges for Managers." New York, NY.: Oxford University Press, 2007, pp. 335.

[3] S. Kim, H. Hwang, and E. Suh, "A Process-based Approach to Knowledge Flow Analysis: A Case Study of a manufacturing Firm," Knowledge and Process Management, vol. 10, pp. 260-276, 2003.

[4] A. Monk and S. Howard, "The Rich Picture: A Tool for Reasoning About Work Context," Interactions, vol. 5, pp. 21-30, 1998.

[5] M. E. Nissen, "An Extended Model of Knowledge-Flow Dynamics," Communications of the Association for Information Systems, vol. 8, pp. 251-266, 2002.

[6] M. Rao, "Knowledge Management Tools and Techniques: Practitioners and Experts Evaluate KM Solutions." Amsterdam: Elsevier, 2005, pp. 438.

[7] O. M. Rodríguez-Elias, A. I. Martínez-García, A. Vizcaíno, J. Favela, and M. Piattini, "Identifying Knowledge Flows in Communities of Practice," in Encyclopedia of Communities of Practice in Information and Knowledge Management, E. Coakes and S. A. Clarke, Eds. Hershey, PA, USA: Idea Group Inc., 2005, pp. 210-217.

[8] O. M. Rodríguez-Elias, A. I. Martínez García, J. Favela, A. Vizcaíno, and J. P. Soto, "Knowledge flow analysis to identify knowledge needs for the design of knowledge management systems and strategies: a methodological approach," presented at 9th ICEIS, Funchal, Madeira - Portugal, 2007.

[9] W. Scholl, C. König, B. Meyer, and P. Heisig, "The future of knowledge management: an international delphi study," Journal of Knowledge Management, vol. 8, pp. 19-35, 2004.

[10] R. Woitsch and D. Karagiannis, "Process-oriented Knowledge Management Systems based on KM-Services: The PROMOTE Approach," International Journal of Intelligent Systems in Accounting, Finance & Management, vol. 11, pp. 253-267, 2003.

[11] F. D. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," MIS Quarterly, 1989, 13(3), pp. 319-340.

Performance: A Longitudinal Study

Nenad Stankovic
University of Aizu
Aizu-Wakamatsu City
Fukushima, 965-8580
Japan

ABSTRACT

Performance is an important nonfunctional requirement of any software system. In this paper we describe our findings from a one year longitudinal case study of the problems identified in two related commercial projects. We divide those problems according to the Pareto principle. The evidence has revealed that a smaller number of modules affected the performance significantly. That finding raises questions about our ability to chart a course and maintain it at both system and subsystem level, and to recognize problems while their resolution is feasible.

On these projects we have also learned that noncritical errors could affect performance significantly. This is largely due to the new technologies that were introduced with no previous in-house experience. But, they could also be attributed to a possible lack of accountability and weak participation in the control process. Noncritical errors may not appear hard to fix, but their number and small individual contribution towards a better performance result when fixed makes their detection, implementation, and testing labor intensive.

1. INTRODUCTION

Software quality has been an ongoing topic of interest in academia and IT industry alike. Software performance has been discussed in a number of forums, such as PDS, IEEE TSE, ICSE, etc. However, the results have been presented as a summary of the raw faults data in a postmortem analysis. Also, due to frequent technological advances and increasingly more challenging applications any past experience and benchmarks becomes harder to correlate to and apply. Still, lessons learned from similar endeavors and technologies should be gathered to enable broader improvement.

Software performance must be constantly monitored [9]. The activities to achieve the required quality can be formulated as these three basic steps:

- Steps taken before implementation,
- Steps taken during implementation, and
- Steps take after implementation.

When developing a software system, we should iterate back and forth among these three steps. In principle, software process also follows these steps, and the steps involved in getting software to perform also breaks down into them. Initial performance evaluation and modeling serves to establish whether our nonfunctional requirements are achievable and to guide our decisions regarding design and software reuse. These can be accomplished either by running a simple test on a raw component, or by undertaking a more formal approach (e.g., [6]]). Also, there is a growing body of performance (anti)patterns (e.g., [2]). Towards the end of development a performance test is undertaken to verify that the performance goals have been met. Between these two steps, however, there is a long period when many decisions are made and must be verified on their merits as part of the process.

While working on a new software system the developers must overcome two major sources of difficulties. Firstly, they must understand the environment in which future system will operate. This is often referred to as application domain and though they do not have to be or become experts it is important to understand the basic concepts and requirements as to design and implement a quality solution. Thus, a system that is not reliable or does not perform is equally unacceptable. Secondly, software development process translates into multiple phases, activities, tasks, and profiles of participants necessary to identify and define a system as a sequence of models that addresses the end user's problem. Software engineering is a modeling activity as it deals with complexity by focusing, at different points in time, only on those details that are relevant and abstract away everything else. In particular, software engineers need to understand the system they could build, the technologies they could use, and the timeframe, to assess different solutions and tradeoffs thereof. These models and artifacts that accompany them cannot always be precisely verified and their accuracy is largely experiential. Software architecture is also influenced by business and social forces from multiple stakeholders [10]. All these are especially true in early project phases when, unfortunately, many important decisions must be made with little evidence to back them up. As these can easily turn into a complex set of issues, here we are primarily interested in performance aspects that we follow through a number of examples.

The analysis presented in this paper is based on two industrial projects that produced two distributed software systems of 650 KLOC (i.e., the smaller project) and 800 KLOC (i.e., the larger project). The systems are implemented in Java and J2EE, and they share some architectural, design, and implementation details. Our analysis is based on a static analysis of the software architecture, design artifacts, and code. It is supported by profiling tools (e.g., OptimizeIt by Borland, and top) and a proprietary dynamic memory code scanner. Multiple commercial load generating tools are used (e.g., LoadRunner [8], Hammer [7]) as well. These help in collecting the multiple metrics [1] that are used to identify problems, and quantify and assess their impact on the performance to the extent possible. In particular, we present a range of results and examine the extent to which they support the following hypothesis:

- A small number off classes is responsible for the major (i.e., critical) performance problems.

- Software performance problems can also be attributed to deficiencies in the staff background.
- Software development process is largely responsible for consistency, and accountability of each participant.
- Similar to scale and distribution of other faults, software projects executed in similar environments share similar performance problems.

For the studied systems we provide evidence for and against these hypotheses. This study is based on two projects only, but we believe that the number and background of the participants works in favor of a more general conclusion than an isolated incident. We find that the first hypothesis holds for the larger project, while is very weak for the smaller. However, these raise a question regarding detection and resolution of performance problems. We describe in detail some of the identified problems and classify them as critical and noncritical with regard to the Pareto rule.

Many factors affect creation, detection, and resolution of performance problems. They can be described as technical (e.g., availability of right tools) and nontechnical (e.g., knowledge, motivation, process). Still, the second and third hypotheses hold. In support, in the section that follows we profile the company. We believe that the software engineers are not only product of their own work, but also of the environment in which they work. Much to our surprise, the forth hypothesis also holds that leads us to conclude that the second, third and forth hypotheses are related. Yet, more data from other companies and projects should be collected to get a better understanding and more confidence in the result.

2. COMPANY

The two projects presented in this paper are first of their kind for this company, both in terms of technology used (i.e., Java, J2EE, RDBMS, IMAP), and domain (i.e., Internet, Unix). To deal with the lack of skills, the staff was trained and new staff were recruited. We find the staff representative of this segment of IT industry. They come from a number of leading companies, with good credentials. During the project initiation phase the company doubled in size at all organizational levels. The organizational model and development process got improved and new software and hardware were purchased. Table 1 and Table 2 present the background as: years of work experience, total people count, and people count per relevant technical and technology related experience before the training and projects started.

Table 1 Management Profile (A=architect, M=manager, P=project lead, T=team lead)

Role	Work	Total	Java	J2EE	C++	RDB	Unix
	< 2						
P, T	2 – 5	4	1	1	4	1	0
P	6 – 7	3	2	1	3	3	1
M, A	13 – 22	3M, 1A.	2	1	4	2	3

Table 2 Developers Profile (E=engineer, S=senior engineer)

Role	Work	Total	Java	J2EE	C++	RDB	Unix
E	< 2	2	1	1	2	1	0
E, S	2 – 5	23	12	5	23	14	7
S	6 – 10	11	11	2	11	10	6
S	> 10	8	4	1	7	6	3

The staff have object-oriented background, and good knowledge of C++. Many have exposure to distributed computing, although not with multitier (four tiers or more) systems. They also had some exposure to Unix, but their background is mainly in Windows. Similarly, their exposure to RDBMS and networking appears adequate in terms of years of experience and previous projects and there is no shortage in the number of staff who can participate creatively and make competent decisions.

In addition, the requirements were prepared well and remained rather stable for the duration of both projects. The project plans follow a waterfall model. The larger project scheduled only one system integration attempt upon the completion of all preceding tasks, which proved very costly. The schedule of the smaller project was similar, but the project team managed to keep functional faults in check but not performance problems. They remained undetected until after the system integration phase. It is important to mention here that before the projects started, solid initial performance tests against a raw component (e.g., IMAP server, RDBMS) or a library (e.g., Java Advanced Imaging) were conduced and produced promising results.

Both schedules were perceived as realistic. The involved were comfortable with their tasks, workload, roles, and organization. All these got eventually confirmed, as the day of completion was narrowly missed. However, the road from there to a product launch proved difficult. The reasons could be attributed to the nonfunctional requirements, i.e., performance and dependability, as they proved interdependent.

Finally, the larger system was predominantly staffed with newly recruited employees that nominally had the background that was required. Some were recruited for a specific role (i.e., architect, database, project leader, or manager). Staff on the smaller project lacked Java and Unix skills, but have some RDBMS. However they have on average 2 years of working together as a team on a C++ / Windows project.

3. CRITICAL PROBLEMS

The 20/80 rule of Pareto, applied to software system performance, stipulates that 20% of code is responsible for 80% of performance problems. Thus, we should focus on those problems that have the greatest potential for reducing the problems. The benefit of this approach is evident in that we achieve most by changing least. The problem is found in identifying the hot spots (or avoiding them) and elaborating on solutions, and the risk in their fixing, all of which can become too hard to overcome.

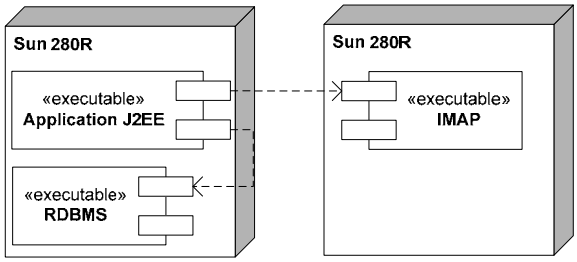


Figure 1 Larger system

Many critical problems described here were created early in the design process and remained undetected until an integration took place. These are multitier systems that run on Sun 280R machines. Each machine has two CPU units and 2GB of RAM. They are connected by a local 100 Mbps Ethernet. These two applications interact with other applications that are not relevant in this context. In this presentation, for the larger project we focus only on the two middle tiers. The deployment diagram in Figure 1 shows two Sun machines with the participating processes of the larger system assigned to each. The smaller system (Figure 2) is the J2EE server that was initially split in two processes: a J2EE process and a Java frontend process that talk via RMI. This architecture was eventually challenged by the performance consultants and consolidated into a single J2EE process.

3.1 Storage

By selecting a persistent storage strategy when designing a system helps in dealing with other issues of storage management (e.g., concurrency, crash recovery, and transactions). It also contributes to system installation and operational cost, depending on our assessment as to how much processing power is needed or available in each particular model. A problem at this point is the inability to back our decisions with solid data because there is no application server in place to run a realistic performance test. The best we can do is to test the raw storage system and assume that this is the top performance that can be delivered by this model. That will be affected negatively by the application server as it progresses towards completion, because each use case will introduce its performance penalty. A major factor here is our understanding as to how much load and what content is expected in production.

The storage consists of two subsystems, i.e., a RDBMS and an IMAP enabled message store (Figure 1). Based on our assumptions on volume of messaging traffic (i.e., data transfer and level of concurrency) we must decide what performance and capacity management characteristics a physical storage must have to adequately support the IMAP server. In addition, what happens once the disk capacity gets exceeded? Can another disk be simply plugged in or do we have to relocate some folders, and upsize manually the remaining. How much CPU time is needed to maintain folders by removing the deleted or old messages and at what time of day? To answer some of these questions we have conducted additional tests. But this problem should also be approached from another direction. The characteristics of these three servers suggest that the IMAP server is predominantly an IO intensive process, both in terms of disks and interprocess communication, while in the same time does not require much RAM to operate. The interprocess communication is highly unpredictable because neither patterns of user behavior, nor size of messages or their content can be accurately predicted, and there are no historical data. However, it has a pool of persistent communication channels, and the level of concurrency approaches that of the Application J2EE.

On the other hand, the RDBMS is easier to model provided that we have a well defined data model. It requires lots of RAM for caching, and each IP connection is supported by one OS process. The interprocess communication needs are constant in size within a few hundred bytes. We know that for up to 10000 bytes the communication cost is mainly affected by the latency and does not change noticeably with message size. The amount of data per query to be transferred here is well within this range. This flat response time is further supported by a pool of connections. The amount of disk space can be estimated for any number of prospective users. The number of transactions approaches the level of concurrency of the Application J2EE server. Finally, the Application server is a major problem for modeling as there are too many unknowns that cannot be verified as there is no implementation yet, and there is no relevant experience as to what resource requirements can be expected. Based on the use cases and functional requirements, we can conclude that it has no particular demands on hard disks, it benefits from a higher IP bandwidth, and it requires, as the preliminary tests demonstrate, lots of RAM and CPU time for the data conversion algorithms. The RAM usage gets even more important as we increase the level of concurrency within the server.

Given all these facts, we can conclude that a configuration in which the IMAP and the Application server share one host because their characteristics are orthogonal, while the RDBMS occupies the other host is the most promising. Also, we have eliminated the need for networking in Application-to-IMAP server communication. The software architecture team, however, decides that the RDBMS and Application should share the same host. In addition, the DB team claims that the preliminary RDBMS tests have proved that a pool of 100 connections is the optimal. We do not know exactly what tests were conducted and what were the assumptions, but we do know that the required Application level of concurrency was 40 transactions per second, and those did not always access the database. In fact, there was no firm definition as to what a transaction is. Therefore, an end to end transaction could spawn multiple subtransactions (e.g., logging) that were all added to the total.

From the logical DB design it is obvious that it is too fragmented. For example, the user profile uses three tables, one of which has each column indexed. The smaller system adds another three user tables, for rules, conditions, and actions. While the latter three are the result of a normalization taken too far, the rationale for the former three cannot be easily explained. Therefore, to create the user profile requires six queries (and J2EE beans), four of which are wild card and two of which are recursive. One table, that stores parameterized data, requires that all columns are indexed. The fact that the RDBMS share the same host with the Application does not make the response time acceptable. A production version of the larger system proved that a pool of 20 persistent connections was sufficient because each query against a loaded DB completes in about 400 milliseconds which meets the requirement of 40 transactions per second, but does not leave enough time for the Application to process messages. However, the 100 was strongly defended.

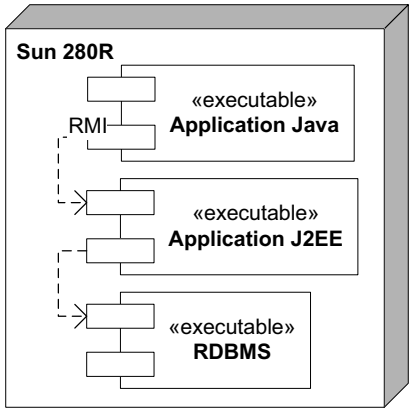


Figure 2 Smaller system

The IMAP work and the RDBMS work were done in isolation which serves better the DB team because they had one client less to serve. IMAP team encountered problems because the IMAP server did not provide the functionality to implement some of the requirements. For example, depending on the class of service each user is assigned the maximum available album size that must be enforced. The class of service is stored in the DB, but the current album size is not and must be calculated each time a message is received. The size of a message is stored in the message header, and the message is stored to a file. As a result, all messages (i.e., files) in the album must be accessed to retrieve the size and add them all up. In a production setup that caused the Application become overloaded with spurious exceptions due to timeouts once the album content exceeded a thousand or so messages.

3.2 Java

These two projects are not challenging because of difficult algorithms but because of the number of services they provide. Although there are no obvious candidates that fall into this category we can still provide a few examples:

- Each system maintains a database table of configuration parameters (~400 per system) that can be updated via a browser based user interface. An event is generated upon change that informs the system to update the table (and its in-memory cache if any). This can be achieved either by executing a query or directly reusing the event. Both approaches are acceptable, even though the latter is much more efficient. A problem with the former is that data must be parsed which makes it even more computationally expensive as our performance profiling has established. The smaller project team leader decided not to maintain a cache but, instead, to execute a query each time a user profile is queried for.
- The aforementioned two process division of the smaller system Application process (Figure 2) requires a RMI interface between the processes. RMI is useful and transparent, but is also known to be very expensive due to the serialization of complex Java objects. Due to garbage collection Java objects are not linear (except for unary native attributes) but are graphs of classes. To make the serialization generic, it is impossible to know how much contiguous memory a graph would require for serialization. Since this RMI channel is the entry point into the system, and the content to be serialized is random in size and complexity, it became a serious bottleneck. The reason for this approach was that the engineers did not know that an applet could be used instead of the J2EE servlet to open a TCP/IP port for incoming traffic.
- To establish whether a substring can be found in a string uses two standard library methods from the Java String class. First a lowercase conversion is performed on the target by making a copy, followed by a check for an occurrence of the substring. This is used by routing rules. There could be up to 20 rules, and each rule is checked separately. The impact on performance is huge as an entire document could be searched through. It is much better to perform a direct case insensitive comparison of both strings, and that code can be copied from JDK. (Management decided that risk appeared too high to implement a searching of multiple related rules concurrently.)

3.3 Pareto

The larger project demonstrates a rather regular Pareto of 17% of all the classes hat can be described as critical. The smaller project is not a typical Pareto system because we can classify up to 35% of the classes as critical. The problem here is not only due to hypothesis 2 but mainly due to its structure. This system has two entry points and two exit points that are of interest for performance. The call stack from message to message is very uniform, which means that most of the code is a candidate to become critical as it gets executed almost if not every time a transaction to process a message is started.

There is another problem with this categorization and that is due to a fact that the critical problems have been identified, but only a few (and those mentioned above) were indeed fixed and those were not storage related. Therefore it is impossible to precisely determine how much impact on the performance of the system as a whole they had or what kind of improvement could have been achieved. As mentioned before little attention was paid to performance after the implementation started and before each system was integrated. With all the coding completed and a lot of unplanned time spent in fixing implementation faults, risk became unacceptable to undertake more rounds of changes. However, we have achieved an order of magnitude better performance on a pilot version of the smaller system that has, apart from the other changes mentioned above, consolidated the mentioned 6 database tables into 3 with no recursion, among others.

These raise another question regarding merits of identification of critical errors. If not caught on time there is little likelihood that they will be fixed because of risk. This suggests that performance consultants will more likely be focused on fixing the noncritical errors towards accomplishing the performance goals than attacking major architectural and implementation issues. These projects certainly followed that route.

4. NONCRITICAL PROBLEMS

Java is a C/C++ look-alike language, but it has quite a different runtime behavior that must be understood and programmed for. Java makes use of a garbage collector to reclaim dynamically

allocated objects and because of that suffers from occasional interruption of service while the garbage collector runs [3]. Garbage collectors provide different modes of operation to deal with this problem, but here we are interested in programming techniques and choices to minimize the impact of garbage collection on performance by reducing the number of transient objects during a method invocation. Programming problems like these are not regarded as critical faults but given the size and nature of this application they were responsible for a large portion of the performance degradation.
Our goal with resolving the noncritical performance problems is to stabilize the process memory image at runtime. We found that for each byte of input 3 to 4 orders of magnitude more bytes were needed to produce a byte of output.

4.1 Common Problems

To concatenate two or more Strings it is more efficient to use a StringBuffer or StringBuilder than the + operator. In fact, this solution is not that simple, and that leads to many programming errors. We have to examine the StringBuffer class to understand its attributes and behavior. The concatenation of two Strings via the + operator creates a new String object, while the StringBuffer approach creates a StringBuffer and requires a new String. It is also important to estimate how long the result is. The default length of a StringBuffer is 16 characters, and we do not want to reallocate the internal *char[]* buffer due to insufficient capacity and copy from the old into a new array. A StringBuffer is reusable because it can be resized, overwritten, and deleted either partially or completely. Many StringBuffer methods can be concatenated, so that the compiler can directly reuse the same object reference that is already loaded in the register.

To get a better understanding on the density of these problems and potential for performance improvements we selected 200 classes in total from 20 engineers. It took us two person days to analyze the code and some results are presented in Figure 3.

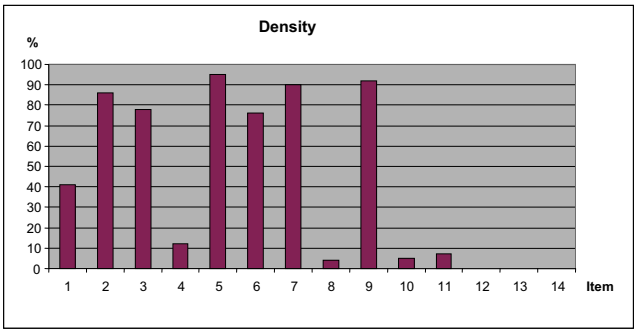


Figure 3 Density

The legend to Figure 3 as per Item is as follows:

1. Dynamic allocation of String constants
2. String concatenation with + operator
3. Dynamic allocation of numeric constants
4. High level logging
5. Not closing sockets
6. Exceptions thrown between local methods upon method invocation
7. Exceptions thrown between classes upon method invocation
8. Catching generic exceptions instead of specific when no resolution is required
9. Replacing native exceptions with domain specific
10. Accessing private public attributes via final methods
11. Intra class nonfunctional integration - passing/reusing buffers for transient data from caller to callee
12. Inter class nonfunctional integration - passing buffers for transient data from caller to callee
13. Inter class functional integration - passing reusable buffer(s) to store result from caller to callee
14. Customize Java core classes for efficiency

These deficiencies may appear simple to resolve but, when multiple classes and methods are involved, they are not. Many are hard to measure and categorize, and their individual impact on performance is disproportionally small to the amount of work required to fix them. To fix these problems, a list of potential improvements was first built and scoped, and then reviewed by the senior staff for a go-no-go decision. Given that our runtime image comprises hundreds of thousands of dynamic objects made the whole process slow and tedious. (As we mentioned before, most of the critical 20% have been declared off limits.)

The initial code and design reviews could spot these flaws but, being performed by peers, they generally failed to do so for the same reason. (Code reviews have been regular and the process was formalized.) Still, we find that peer inspections tend to degenerate into comments on style and first order semantic issues. This stands in contrast to the reported findings (e.g., [2]) and commonly shared believes. We believe, though, these teams are representative for a semidetached or embedded project both in size, age, and experience.

Item 4 is a serious problem due to the design of the Logger. The Logger is a deeply nested hierarchy of classes that does not apply a high level filtering of information for logging. Instead, if a programmer does not check the logging level before attempting to create a log, the logger will do it at the lowest level. This problem is compounded by an excessive logging for tracing. Also, the parameters are passed as an array of Strings and numbers of an arbitrary size.

Items 11 through 14 are positive (i.e., desired outcome) and they range from 0% to 7%. While the issues that fall under 11 and 14 could become candidates for improvements, the other two are not trivial and point to the understanding as to how well a system is integrated in those aspects that are not part of the functional specification. They suggest that these engineers have not discussed issues other than what is required to produce a result.

4.2 More Involved Java Example

This example falls under Item 14 that has a density of 0%. It extends the findings in the String example above but, in order to improve it, requires a bit more coding in addition to the default behavior. The program reads from a stream by converting it into a String, one per line (i.e., 80 characters) of input. Then, it creates a StringTokenizer to remove the control characters from the String. Each token is appended as a byte array to a ByteArrayOutput-Stream (BAOS) of default size (32 bytes initially). Finally, the BAOS is trimmed before being passed down the call stack.

This algorithm was implemented such that it uses whatever functionality is provided by these classes. It is easy to read and understand. However, each benchmark message of 1000 characters creates 13 transient Strings of input, 13 tokenizers, and two Strings and two byte arrays per each control sequence found (on average one per line), and two BAOS of ~1000 bytes each. These figures include only those objects that are visible to and manipulated by the programmer.

In reality, all these transient objects can be replaced by a single BAOS (albeit slightly modified) and a simple filter as the switch block with three cases. We read into a BAOS buffer, and then shift left by one whenever a control sequence is found to overwrite it. The BAOS could be passed down the stack as is to avoid a copy and reused since it knows how many bytes are valid.

5. REMARKS

In a popular book on refactoring (i.e., [5]) the importance of disciplined and one-step at a time approach to code improvements is emphasized. Unfortunately, performance issue and goals were not addressed. Thus, even some of the examples ended up in a code that is performance-wise inferior to the original. In that respect, the code improvements discussed here do not necessarily follow the rationale for refactoring as presented in the literature.

Returning to these applications, the mentioned problems affected the performance so that it was only a fraction of what was required. The critical problems had not been addressed (or identified) by the staff. Even so, they would most likely be rejected by management as being too risky and costly, as we have learned later. Only when the consultants joined the projects they started identifying and investigating critical problems. It is interesting to notice that, on the other hand, many code improvements in terms of restructuring, consolidation, and reuse were proposed and implemented. Unfortunately, the performance improvements, as a result of that work, still remained sketchy, which leads us to believe that performance as a goal in projects or training is not a priority.

We can conclude that Hypothesis 1 holds, even though the smaller system is significantly over the limit of 20%. Hypothesis 2 has deeper implications if we can generalize it to other projects. The staff on these projects come from reputable companies and their background that should facilitate a positive outcome, which failed to eventuate. While some of the performance problems can be attributed to politics and personal preferences it is still troubling to see that they remained undetected until a system integration. In that respect Hypothesis 3 is supported by the fact that the smaller system was completed with lesser faults and performance problems and managed to achieve substantial (5 fold) performance improvement before being launched. This is counterintuitive when considering that most of those engineers had no previous exposure to Java. However, their strength was in the better project management and time on past projects to improve their teamwork. Finally, Hypothesis 4 also appears counterintuitive if we only consider the background of the staff. On the other hand Figure 3 clearly suggests that in fact the density of items under investigation was not in favor of larger project, but was rather even in most aspects. We can find an explanation for this outcome given that both projects were developed in the same time, and shared many decisions and information as deemed appropriate or when required. Hypothesis 3 suggests that what was a weakness of one team became a strength of the other.

6. REFERENCES

[1] Armour, P.G. Beware of Counting Lines of Code. *Communications of the ACM*, 47(3), 2004, pp.21-24

[2] Boehm, B., and Basili, V.R. Software Defect Reduction Top 10 List. *IEEE Computer*, 34(1), 2001, pp. 135-137.

[3] Blackburn, S. M., Cheng, P., and McKinley, K. S. Myths and Realities: The Performance Impact of Garbage Collection, *In Proc. of the Conference on Measurement & Modeling Comp. Sys.*, New York, NY, 2004, pp. 25-36.

[4] Brown, W.J., Malveau, R.C., McCormick III, H.W., and Mowbray, T.J. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley and Sons Inc., New York, NY, 1998.

[5] Fowler, M., *Refactoring Improving the Design of Existing Code*, Addison Wesley Longman Inc., Reading, MA, 1999.

[6] Menasce, D.A., and Gomaa, H. A Method for Design and Performance Modeling of Client/Server Systems. *IEEE TSE*, 26(11), November 2000, pp.1066-1085.

[7] Empirix, www.empirix.com

[8] LoadRunner, www.mercury.com

[9] Smith, C.U., *Performance Engineering of Software Systems*, Addison Wesley, 1990.

[10] Weinstock, C.B., et al. *Steps in an Architecture Tradeoff Analysis Method: Quality Attribute Models and Analysis*. CMU/SEI-97-TR-029, Pittsburgh, PA, 1998.

A Formal Approach for Translating a SAM Architecture to PROMELA

Gonzalo Argote-Garcia, Peter J. Clarke and Xudong He
School of Computing and Information Sciences
Florida International University
{gargo001,clarkep,hex}@cis.fiu.edu

Yujian Fu
Department of Computer Science
Alabama A&M University
yujian.fu@aamu.edu

Leyuan Shi
Department of Industrial and Systems Engineering
University of Wisconsin at Madison
leyuan@engr.wisc.edu

Abstract

Quality assurance is recognized as a critical aspect in software construction. SAM is a formal software architecture description model that combines Petri Nets and Temporal Logic. PROMELA is the language used in the Spin model checker. This paper presents an approach to translate a restricted SAM model to a PROMELA program, enabling the model checking of the SAM model. We define the translation and show its correctness in terms of completeness and consistency. Completeness establishes that a SAM model maps all of its elements to PROMELA ones; whereas, consistency defines that an execution of a SAM model has a corresponding execution in a PROMELA program. The translation is also implemented as part of our software tool supporting SAM. Some aspects of the tool are discussed.

Keywords*: SAM Architecture, PrT Nets, Spin, Model Checking, Model Translation Validation, Tool Support.*

1. Introduction

Software Quality Assurance is of great importance in the development of software systems. From design to implementation, there is a need to ensure the system satisfies desired properties. One way to help in achieving this, is by defining a formal architecture model of the system and proving its correctness with respect to the desired properties. This is critical, given the huge impact that software architecture has on the software system.

The Software Architecture Model (SAM) Framework [6] supports the formal modeling of software architectures. Based on Petri Nets and Temporal Logic, SAM is used to define the structure and behavior of a software architecture, and the properties it needs to satisfy. The resulting SAM model can then be analyzed using various formal methods techniques and tools [6].

There are several techniques to formally analyze software systems. Model checking is one successful technique used in the verification of finite state systems. A prominent model checking tool is Spin [9]. In Spin, a verification process is done by executing a model written in PROMELA (Spin's input language), and checking it against a property.

In this paper, we present an approach to translate a restricted SAM model to a PROMELA program, enabling the use of the model checker Spin to verify properties for the model. For the translation to be considered correct, the resulting PROMELA code has to reflect every element in the SAM model (completeness) and it has to preserve the semantics of the model (consistency). For this to happen, we restrict the kind of SAM models that can be verified.

This work is relevant to the "Grand Challenge" proposal [8], in which Hoare highlighted the importance of incorporating techniques that have been proved successful in the software development process to ensure software correctness. Our work is also part of *"A Framework for Ensuring System Dependability from Design to Implementation"* proposed in [5], which incorporates analysis techniques such as model checking, testing and runtime verification to assure the quality of software systems. That framework is supported by a modeling and analysis tool. We extended the tool with the implementation of the translation approach, resulting in the automatic generation of PROMELA code for a SAM model. Spin can then be used to execute the PROMELA program to verify desired properties.

The rest of the paper is organized as follows. We provide a brief background in Section 2. In Section 3, the restricted SAM model and the translation approach are explained. The translation implementation is discussed in Section 4. Section 5 presents the case study, and Section 6 discusses

related work. Finally, Section 7 states our conclusions.

2. Background

A brief introduction to SAM and Spin is provided.

2.1. Software Architecture Model (SAM)

A SAM model consists of a set of compositions $C = \{C_1, C_2, ..., C_k\}$ with a top level composition $C_l \in C$ representing the top level design. Each composition $C_i = (Cm_i, Cn_i, Cs_i)$ consists of a set of Cm_i components, a set Cn_i of connectors, and a set Cs_i of composition constraints. A component or connector $C_{ij} \in Cm_i \cup Cn_i$ is non-elementary if it is refined by a lower level composition in C; otherwise, it is elementary. Each $C_{ij} = (S_{ij}, B_{ij})$ has a property specification S_{ij} and a behavior model B_{ij}. First order Linear Temporal Logic and Predicate Transition Nets are used to define the properties and the behavior respectively. The property specification and behavior model for a non-elementary C_{ij} is obtained by merging the behaviors and specifications of the components and connectors of the composition mapped to it (see [6]). In our approach, each non-elementary component/connector is replaced by the corresponding components and connectors in the composition it maps to; as a result, the top level composition will only contain elementary components and connectors.

Predicate Transition Nets (PrT Nets). A PrT Net [3] consists of a net structure (P, T, F), an algebraic specification (S, Op, Eq) and a net inscription (φ, L, R, M_0). The most important aspects to note are that each $p \in P$, where P is the set of predicates (places), is mapped to a sort $s \in S$ ($\varphi(p) = s$) and contains tokens that are ground terms of its corresponding sort s. T is the set of transitions and R defines for each transition $t \in T$ its precondition and postcondition expressed as first order logic formulas. The arcs in F connect places and transitions, and have labels defined by L which are used in the pre and post conditions in transitions. A transition is enabled if there is a substitution for the variables in the incoming arcs that satisfies the transition's precondition. The substitution is achieved by assigning tokens in the corresponding place to each variable. A transition is fired if it is enabled, and the postcondition is then satisfied. Finally M_0 represent the initial marking, i.e. the initial tokens contained in the places. For a more detailed presentation on PrT Nets, refer to [3].

Linear Temporal Logic (LTL). LTL [10] has been widely used to specify properties for software systems. In the SAM framework, property specifications for components/connectors and constraints for compositions are defined in first order LTL. A first order LTL formula contains predicates as terms and can contain universal quantifiers. Since the model checker Spin verifies properties defined in propositional LTL, in our approach the properties and constraints for SAM models are modified so that the LTL verification power of Spin can be applied to them.

2.2. Spin and PROMELA

Spin is a well known model checker used in the verification of finite state systems. PROMELA is its input language. PROMELA's emphasis is on models that describe the coordination and synchronization aspects of a distributed system and not on the computational aspects of it [9]. PROMELA is different to common procedural programming languages, such as C, in that it includes features intended to model distributed systems while lacking some other features found in programming languages. Nevertheless, we can still include complex computations in a PROMELA model by using the embedded C-code feature. In our case this is necessary since for transition firing, both testing the enabledness of a transition and firing it can consist of complex computations. For instance, tokens for a place are stored in an array, and we need to iterate through the array to find the appropriate tokens to be used in the firing of a transition.

3. Translation

In this section, we first introduce a restricted version of SAM and PrT Net models that allows us to provide a sound translation approach. Next, the translation approach is defined and we present the mapping between elements in the SAM model and the PROMELA program. In addition, a discussion on the correctness of the translation is provided. Finally, we describe alternative translation approaches.

3.1. Restricted SAM and PrT Net Models

Components and connectors connect through ports, but in the hierarchical view of a SAM model, a port can be mapped to multiple ports in the lower layers. For this reason, we define the following restriction:

One-to-one port mapping between compositions: Given two compositions C_i and C_k, where C_k is the refinement of one component or connector C_{ji} in C_i, a port in C_{ji} can only be mapped to one port in C_k.

In order to be translated to a PROMELA program, a SAM model needs to have a finite state space. The state space of a SAM model is defined by its behavioral model, a PrT net model; hence, the PrT net model needs to have a finite state space. Not only do we restrict the sorts but also limit the number of tokens each predicate (place) in the PrT net can have.

Restrictions on Sorts. We consider the following: (1) finite number of ground terms, (2) real numbers, (3) strings and (4) derived sorts. We elaborate on these aspects next.

(1) *Finite number of ground terms*. The basic sorts for the restricted version of PrT Nets are defined by:

$s_{basic} ::= string|bit|bool|byte|short|int|unsigned$

The number of ground terms for each basic type is considered to be finite.

(2) *Real numbers*. A finite set of real numbers can be mapped to integer values. We assume that this mapping is done explicitly in the model built.

(3) *Strings*. Sort $string$ is restricted to represent a fixed number of strings by mapping it to the $short$ type: each string ground term is mapped to a unique integer number in a sequential fashion (the strings encoding). String operations are reduced to assignment and comparison. Concatenation and others are not available.

(4) *Derived sorts*. Derived sorts are of the form $\wp(s)$ (powerset) and $s_1 \times s_2 \ldots \times s_n$ (cross-product), with $s, s_1, ..., s_n$ being sorts. Since basic sorts have a finite number of ground terms, so do the derived ones. However, that number might explode. For instance, sort $short$ has 65536 ground terms, but derived sort $\wp(short)$ accounts for 2^{65536} possible ones. We limit the number of elements each subset can contain.

With the above considerations, the set S of sorts for a PrT Net contains elements defined by:

$$s ::= s_{basic} \mid s(\times s)^+ \mid \wp(s)$$

Bounded Nets. Each $p \in P$, with P being the set of places in the behavioral model, is bounded.

3.2. Translation Approach

First, a flattened version of the SAM model is created, and next the actual translation takes place (See Figure 1).

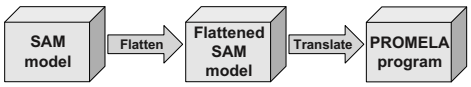


Figure 1. Translation approach.

3.2.1. Flattening SAM

Flattened version of a SAM Model. The set of compositions $C = \{C_1, C_2, ..., C_k\}$ for a SAM model, is reduced to a set $C' = \{C_l\}$ containing one composition. For each non-elementary component/connector C_{ji} in composition C_i that is refined by composition C_k, we do the following:

(1) C_{ji} is replaced by the components and connectors Cm_k and Cn_k in C_k.

(2) The property specification S_{ji} for C_{ji} is added to the set of constraints Cs_i for C_i.

(3) The set of constraints Cs_k in C_k is added to the set of constraints Cs_i for C_i.

Having the flattened version of a SAM model, we obtain its integrated behavioral model next.

Integrated Behavioral Model. Given a SAM model consisting of only one composition $C_i = (Cm_i, Cn_i, Cs_i)$, its integrated behavioral model B_i is created by combining the behavioral models of each $C_{ji} \in Cm_i \cup Cn_i$ ($C_{ji} = (B_{ji}, S_{ji})$), i.e. $B_i = \bigcup_j B_{ji}$.

3.2.2. Translating Flattened SAM to PROMELA

In the translated PROMELA program, we use the embedded C code feature of PROMELA to define functions implementing various notions in the SAM model. For example, a function that adds tokens to a place is defined. In Figure 2 we show a table that gives a snapshot of a PROMELA program. Each row represents a section in the program and we briefly present each one next.

Section	PROMELA code example	Relation to SAM
1	#define BOUND_P1 maxP1	Bound value for place P1.

2	typedef PSET{
	  string set[max];
	  short num;
	};
		Definition of powerset sort PSET to be used to define sets of strings.

3	c_code{
	  int is_equal_PSET(PSET l, PSET r){
	  // return 1 if equal, 0 otherwise
	  }
	}
		Operations on sorts. Here, equality testing for two elements of type PSET is defined.

4	PSET v_Port1[BOUND_Port1] short num_Port1	Port1 is a place related to a port.

5	c_code{
	  void add_P1(...){ ... }
	  void remove_P1(...){ ... }
	}
		Add/remove tokens to/from P1 when a transition having P1 as part of its pre/post set fires.

6	c_code{
	  int is_enabled_T1(...){ ... }
	  void fire_T1(...){ ... }
	}
		Testing the enabledness and firing a transition T1.

7	proctype Comp{
	  // initial marking
	  // wait for the other procs to initialize
	  do
	  :: atomic{ c_expr{is_enabled_T1(...)}
	  ->c_code{fire_T1(...)}}
	  od
	}
		Component Comp. Initial marking: Comp process initializes its state, waits for the other processes to initialize. Behavior execution: after initialization, Comp fires its transitions if they are enabled.

8	init{
	  // initial marking for port places
	  // initialize synchronization constants
	  atomic{
	  run Comp();
	  }
	}
		Initial marking: sets the initial marking for places acting as ports. Executes the Component processes for them to start the execution of the Petri nets.

9	#define p c_expr{P()}
	never{
	  // LTL automaton definition
	}
		The property to be verified.

Figure 2. Translated code overview.

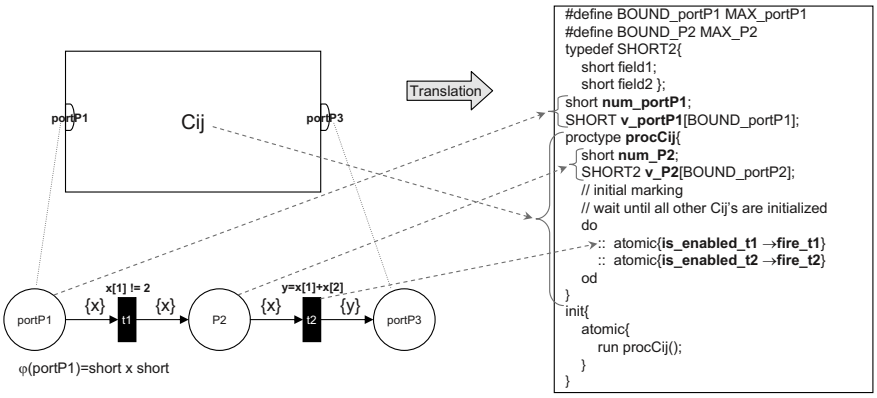


Figure 3. A translation example.

In row 1, constant symbols representing the bounding values for places are defined. Row 2 defines the cross products and power set types and their operations are implemented in row 3 using PROMELA's embedded C code facility. The definition of places acting as ports is done in row 4; those variables are defined globally for two components/connectors to communicate. Also, row 4 contains global synchronization variables used to wait for each process to finish setting the initial marking corresponding to its behavioral model. Row 5 implements code to add/remove tokens to/from places. In row 6 we have embedded C code for testing the enabledness and executing the firing of transitions.

Row 7 is where the $proctype$ definition for each component is placed. This $proctype$ defines variables for the places of its behavioral model that are not related to ports (the places related to ports are defined globally). Code for setting the initial marking is added, and a synchronization point is established to wait for the other processes to set their initial markings. Finally, an infinite loop tests the enabledness and executes the firing of its transitions.

In row 8, process $init$ sets the initial marking for places related to ports, initializes the synchronization variables and starts the execution of the processes. At last, row 9 includes the $never$ claim for the property to check.

In Figure 3 we can see a graphical example showing an outline of the translation approach. Given their significance, we present two aspects of the translation in more detail: the behavior (PrT net) translation and the property/constraint translation.

Behavior Translation. A PrT Net consists of a finite net structure (P, T, F), an algebraic specification (S, Op, Eq) and a net inscription (φ, L, R, M_0). Those elements are closely related; for example, a predicate $p \in P$ has sort $\varphi(p) \in S$, and initial marking $M_0(p)$. All these elements are reflected in the target PROMELA code. We divide the behavior translation process in three parts: sort translation, place translation and transition translation.

(a) **Sort Translation.** Sorts are defined by:
$s ::= s_{basic} \mid s(\times s)^+ \mid \wp(s)$
$s_{basic} ::= string|bit|bool|byte|short|int|unsigned$

Basic sorts. Each basic sort, except $string$, has a PROMELA counterpart with the same name. Sort $string$ is mapped to type $short$. The usual operations for integer and boolean types are available. For sort $string$ only comparison and assignment are allowed.

Cross product. Sort $s = s_1 \times s_2 ... \times s_n$ translates to:

$typedef\ s = \{$
$\quad s_1\ field1;$
$\quad s_2\ field2;$
$\quad ...$
$\quad s_n\ fieldn$
$\};$

Equality and assignment operations are defined. For sort s the prototypes are:

$int\ is\_equal\_s(s\ v1,\ s\ v2)$
$void\ assign\_s(s\ *v1,\ s\ *v2)$

Powerset. Sort $s = \wp(s_1)$ is translated as:

$typedef\ s = \{$
$\quad int\ num;$
$\quad s_1\ set[max\_n];$
$\};$

With max_n the maximum number of elements in a set. The operations are the usual set operations, plus equality and assignment. Some operation prototypes for powerset sort s are:

$int\ is\_equal\_s(s\ v1,\ s\ v2)$
$s\ set\_union\_s(s\ p1,\ s\ p2)$

(b) **Place Translation.** Given a place $p \in P$ the following are defined in PROMELA:

Bounded Values. A constant symbol $BOUND\_p$ is defined to state the maximum number of tokens in p. By default this number is set to 10.

Tokens Storage. Let $s = \varphi(p)$ be the sort of place p. We define two variables for dealing with tokens at p:

$s\ v\_p[BOUND\_p];$
$short\ num\_p;$

Array v_p holds the tokens, $BOUND\_p$ defines the maximum number of tokens, and num_p is the current number of tokens in p ($0 \leq num\_p \leq BOUND\_p$).

Initial Marking. We translate $M_0(p)$ by creating a series of expressions in PROMELA language to initialize each token. For example, assuming $\varphi(p) = s_1 \times s_2 ... \times s_m$, with each s_i being a basic sort, and $|M_0(p)| = n$, we generate:

$num\_p = n;$
$v\_p[0].field1 = val_1;$
...
$v\_p[n-1].fieldm = val_{nm};$

Add, Remove and Test Tokens. For place p, functions add_p and $remove\_p$ are implemented to add and remove tokens. Also, we define function in_place_p to test whether or not a given token is at the place.

(c) **Transition Translation**. We only offer an overview on how a transition $t \in T$ is translated:

Enabledness. Given $p_i \in {}^\bullet t$, testing the enabledness of transition t involves a substitution on the variables in $L(p_i, t)$, with tokens in p_i, and then testing the precondition in $R(t)$. We define a function that given preset ${}^\bullet t$, it either returns 1 if there is a substitution that satisfies its precondition, or 0 otherwise:

$int\ is\_enabled\_t(preset\_t)$

Firing. When t fires, tokens are removed from ${}^\bullet t$ and tokens are added to $t^\bullet$. The function prototype that implements the firing notion for t is:

$void\ fire\_t(preset\_t,\ postset\_t)$

Universal Quantifiers. We add functions to test the truth value of universal quantifiers formulas whenever a transition's $R(t)$ includes them:

$int\ forall\_t(params)$
$int\ exists\_t(params)$

Property and Constraint Translation. Constraints Cs_i for composition C_i and property specification S_{ji} for each component/connector $C_{ji} \in C_i$, are expressed in first order LTL. We briefly mention some aspects on how an LTL formula is translated to PROMELA code. Given an LTL formula, first, all the universal quantifiers affecting predicate terms, i.e., the places in the behavioral model, are removed. Next, each predicate term P in f generates a macro:

$\#define\ p\ c\_expr\{P()\}$

The body of function $P()$ reflects an instantiation of the predicate to a propositional formula. It returns an integer value of 1 or 0, depending on whether the formula is true or not. There are a few approaches on how to select the ground terms for instantiating the predicate, we select the one that considers the tokens in the initial marking as the possible values. Finally, the $never$ claim is generated using property automaton generator facility available in Spin.

3.3. Translation Correctness

We show the completeness and consistency of our translation approach.

Interleaving semantics. *Both for the Behavioral model in SAM and for the translated PROMELA model, the interleaving execution semantics is chosen.*

Since there is no true concurrency in PROMELA, this is an important observation that will allow the two models to be compared. For more on interleaving semantics w.r.t. true concurrency refer to [9].

CLAIM 1 ***(Flattened version correctness)*** *The flattened version of a SAM model respects the original model's behavior and specification.*
PROOF *Follows directly from the flattening procedure.*

We can think of the compositions in a SAM model as different ways to partition it.

3.3.1. Completeness

CLAIM 2 ***(Completeness of the translation)*** *Given a restricted SAM Model, there exists a corresponding PROMELA model that defines all of its elements.*
PROOF *Follows directly from the mappings.*

In the translation process, each of the elements in the SAM model are translated to a PROMELA construct. Two elements to pay special attention to are the property specification and the initial marking. For the first one, we mentioned how to convert a predicate into a proposition. For the second one, we explained how to define a series of expressions to build the initial marking for each component/connector and how all the components/connectors are synchronized before testing the enabledness and firing their transitions.

3.3.2. Consistency

CLAIM 3 ***(Initial Marking consistency)*** *Given a restricted SAM model* sam *and its translated PROMELA program* prom*, the initial marking in the underlying behavioral models in* sam *is consistent with the state* prom *previous to the point where the processes test and fire the translated transitions relations.*

PROOF *In model* sam*, the initial marking of its components and connectors is defined as part of the model itself. In the PROMELA program* prom*, there is a series of steps that make the variables related to the places take the initial values, and no process executes the enabledness testing and firing of transitions before every other one has initialized its variables. In* prom*, there is a synchronization step before the* do::...::od *main loop in each process. This synchronization step waits for a global variable to reach the number of processes that have been initialized:* (init_procs == num_procs)*. Hence, the initial marking in* sam *corresponds to the state in* prom *previous to which each component/connector process starts to execute the enabledness testing and firing of transitions.*

For the semantic consistency claim, we define the notion of abstract execution of a translated PROMELA program.

Translated PROMELA program abstract execution sequence *After initialization, when the initial marking of the model is set, the only executable instructions in the processes are the ones within the do..od construct:*

$proctype\ proc_{ij}()\{$
do
$:: atomic\{is\_enabled\_t_1 \rightarrow fire\_t_1\}$
$:: atomic\{is\_enabled\_t_2 \rightarrow fire\_t_2\}$
...
$:: atomic\{is\_enabled\_t_n \rightarrow fire\_t_n\}$
od
$\}$

An abstract execution is a sequence $\sigma_0 fire\_t_a \sigma_1 fire\_t_b ...$*, where a change of state occurs only when an executable statement* $fire\_t \in \{fire\_t_1, fire\_t_2, ..., fire\_t_n\}$ *to the right of the arrow is executed.*

Since, the enabling and firing are atomic constructs, the firing of transitions in PROMELA has the same meaning as the firings of transitions in the PrT model.

CLAIM 4 ***(Semantic Consistency of a SAM model and its translated PROMELA program)*** *A SAM model* sam *is semantically consistent with its translated PROMELA program* prom*, iff for every execution sequence in* sam *there is a corresponding abstract execution in* prom*.*

PROOF *Follows directly from the definition of abstract execution for a PROMELA model.*

Since in the PROMELA program there is a finer granularity, for instance, when firing a transition there are multiple instructions that need to be combined to realize it, we work on the abstract version of an execution sequence in PROMELA. This abstract sequence is just an aggregate of the different sub steps. When firing a transition these sub steps are part of an aggregate atomic construct, they are uninterrupted and hence can be seen as a single step.

3.4. Discussion

Some alternatives to the translation are presented.

Non-flattened Composition Translation. A non-flattened SAM model m, consists of multiple levels of compositions. One way to translate m to PROMELA is to define each component/connector at different levels as a process. A consequence is that if a component maps to a composition, then it will contain calls to other processes, the processes corresponding to the components/connectors in the refining composition. As a result, a flattened version is preferred.

Transition as process. Each transition in the PrT net model can be defined as a process. Given that we have interleaving semantics for the execution of the PROMELA code, we have the same effect as if the transition code is part of a process. If the transition is picked to be fired, it will fire without interruption. Other reason for not choosing this alternative is that Spin limits the number of processes that can be run, so if we have a model with several transitions, we might exhaust the available processes.

Transition enabling and firing construction. When testing the enabledness of a transition t, another approach is to compute all the substitutions for the variables in the incoming arcs, next we can compute the substitutions that enable the transition, from these enabling substitutions, we can next randomly choose one to be used for the actual firing. For each $p \in {}^\bullet t$, we create a matrix of indexes with $|L(p,t)|$ columns and $(|p|) \div (|p| - |L(p,t)|)$ rows. Next we can do a random selection on the matrices corresponding to the incoming places for a transition. We haven't conducted experiments on this approach as yet.

First Order LTL Expansion. We can expand a formula to include all possible substitutions for the predicates involved in it. For example, given a formula $\forall x \cdot (\Box(P1(x) \rightarrow \diamond P2(x)))$, where $\varphi(P1) = \varphi(P2) = short \times short$, we have this complete enumeration: $\Box(P1(<0,0>) \rightarrow \diamond P2(<0,0>)) \wedge ... \wedge \Box(P1(<0,n>) \rightarrow \diamond P2(<0,n>))...\Box(P1(<n,0>) \rightarrow \diamond P2(<n,0>)) \wedge ... \wedge \Box(P1(<n,n>) \rightarrow \diamond P2(<n,n>))$. Where n is the size of short. So we have n^2 formulas. This provides impractical.

4. Tool Implementation

The translation approach presented in this paper was implemented as part of our modeling and analysis tool for the SAM framework (SAM tool). SAM tool incorporates functionality for the graphical view and creation of SAM models as well as their analysis. We show some aspects of the tool.

Features. SAM tool provides a graphical editor with editing capabilities to build SAM models as well as the PrT net behavioral models for each component and connector. The hierarchical structure is displayed as part of a tree view. We can see a snapshot of the tool editor in Figure 4. It provides the basic capabilities of any graphical editor. However, we are still enhancing the editor with other functionality such as drag and drop within the hierarchical view.

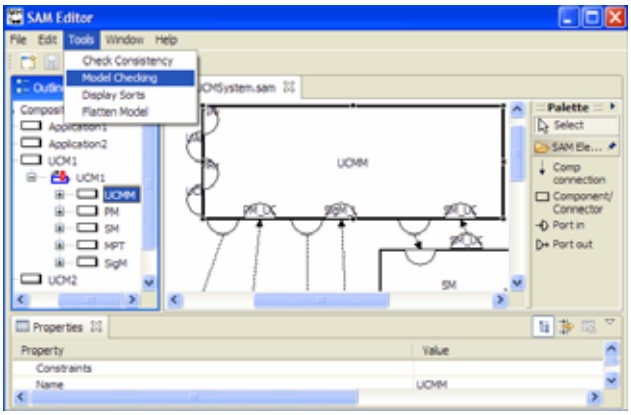


Figure 4. SAM Tool.

Intermediate XML format. A SAM model is serialized and unserialized using the serialization techniques in Java, so the model is saved in binary format. The tool also provides the capability to export and import the SAM model in XML format. This XML format follows a similar layout as the standard PNML (Petri Net Markup Language) format defined for Petri nets. We extended PNML format by adding the notions of compositions, components, connectors and first order LTL formulas.

Translation module. SAM tool has the option to generate the PROMELA code for the current SAM model being edited. The option is available through the menu bar as shown in Figure 4. Since we plan on providing command line tools, we implemented the translation module so that it could be easily isolated from the editor. So, the translation module, first internally exports the current model to XML format, then performs the flattening procedure on the XML model, and next generates the PROMELA code for it following the approach detailed in this paper. This is transparent to the user, since he/she only needs to enter the filename of the target PROMELA program, and a notification dialog is displayed after the translation is completed.

Implementation details. The tool is implemented as an Eclipse RCP application (Rich Client Platform application). It uses GEF (Graphical Editing Framework) for the graphical editing of objects. GEF provides and enforces several design pattern constructs, which are intended to make the maintenance particularly easier. For more details on Eclipse RCP and GEF refer to [12].

One other module of interest is the formula editor, which is written using Java Swing. Since Java Swing and Eclipse RCP define and use different libraries, we had to incorporate the formula editor to the Eclipse RCP Application. The formula editor contains two submodules, a FOL (first order logic) parser and a FO-LTL (first order LTL) one. Both are implemented based on the well-known Java-CUP parser generator. As a result we are able to tell the user whenever a formula was properly written or not.

5. Case Study

We applied our approach to two non-trivial examples, one an architecture for a communications virtual machine and the other a communications protocol. We used our SAM modeling and analysis tool (SAM tool) to model and translate the examples to PROMELA programs. Spin was then used to verify some properties.

The first case study is on the formal model of the Unified Communication Machine (UCM) presented in [13]. We defined a system with 6 top level components from which 2 were UCMs at different sites. One of the UCMs was refined into 5 components, which were completely specified. One of the components, the UCMM (UCM manager), was of special interest, and some properties were verified for it. The integrated behavioral model (PrT net) consisted of 63 places, 140 transitions and 476 arcs. Our software tool automatically generated the PROMELA model for it. The size of the verification code generated for this case study was around 10,000 lines of code. This case study was based on the formal modeling example presented in [11].

The second model we applied our technique to is the well-known Alternating Bit Protocol. This model is much simpler than the former one, it consists of only 3 components: the sender, the channel and the receiver. The main property we verified was a liveness property of the form: $\forall x \cdot (\Box(Send(x) \rightarrow \diamond Recv(x)))$. We instantiated the formula to propositional ones depending on the tokens available in the initial marking. For example, the initial marking for $Send$ was $M_0(Send) = \{\text{“}first\text{”}, \text{“}second\text{”}\}$, and we first verified a formula of the form $\Box(Send(\text{“}first\text{”}) \rightarrow \diamond Recv(\text{“}first\text{”}))$ and next another formula of the form $\Box(Send(\text{“}second\text{”}) \rightarrow \diamond Recv(\text{“}second\text{”}))$. However, if

$|M_0(Send)|$ is big, then the latter is impractical.

One observation, resulting from the experiments performed, is that for a non-satisfiable liveness property, Spin would not directly state that it was not satisfied. It would report that an acceptance cycle was encountered. Hence, when verifying a liveness property we must include the flag for acceptance cycles in the Spin verification environment.

6. Related Work

There has been work done in both, model checking of SAM models and translation of them into other models. For example, a translation from SAM to Java was proposed in [1], with the added capability of a runtime checker. In our case, we are not interesting in translating to some executable form that can be later refined to implement a working system, rather we want to assure the properties a SAM architecture defines. In terms of model checking, SMV was used in [7] to verify properties defined in CTL (Computational Tree Logic); in our case we use LTL.

Gannod et. al. [2] used Spin to verify Petri net models using the DOME tool. They integrated DOME with Spin to provide a modeling analysis environment. Also, Grahlmann et. al. [4] integrated Spin into the PEP tool (Programming Environment based on Petri Nets). We also did something similar by combining our SAM modeling and analysis tool with Spin. However, in our case we used PrT nets which provide the behavioral model of architectural components and connectors, and we used Spin to verify properties at the architectural level.

7. Conclusions

Assuring the quality of software systems is a big challenge. There are well known techniques to analyze software models. One prominent technique is model checking; however, its applicability is limited. Even at the architectural level some trade offs need to be taken when dealing with model checking techniques. In our case, we had to restrict the kind of SAM models to be model checked in Spin. Furthermore, we had to limit the properties to verify, from first order LTL to propositional LTL.

By applying our translation approach, we were able to successfully prove properties for SAM models using our modeling and analysis tool and the Spin model checker. Hence, extending our baggage of tools and techniques to verify SAM models, all aimed at contributing to the general framework presented in [5], to ensure system dependability from design to implementation.

Acknowledgments. This research was supported by NSF grant HRD-0317692 and NSF grant IIP-0534428.

References

[1] Y. Fu, Z. Dong, G. Argote-Garcia, L. Shi, and X. He. An Approach to Validating Translation Correctness from SAM to Java. In *SEKE*, pages 45–. Knowledge Systems Institute Graduate School, 2007.

[2] G. C. Gannod and S. Gupta. An Automated Tool for Analyzing Petri Nets Using SPIN. In *ASE '01*, page 404, Washington, DC, USA, 2001. IEEE Computer Society.

[3] H. J. Genrich. Predicate/Transition Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets*, volume 254 of *Lecture Notes in Computer Science*, pages 207–247. Springer, 1986.

[4] B. Grahlmann and C. Pohl. Profiting from Spin in PEP. In *SPIN'98 Workshop*, 1998.

[5] X. He. A Framework for Ensuring System Dependability from Design to Implementation. In U. Ultes-Nitsche, J. C. Augusto, and J. Barjis, editors, *MSVVEIS*. INSTICC Press INSTICC Press, 2005.

[6] X. He and Y. Deng. A Framework for Developing and Analyzing Software Architecture Specifications in SAM. *The Computer Journal*, vol.45(no.1):111–128, 2002.

[7] X. He, J. Ding, and Y. Deng. Model checking software architecture specifications in SAM. In *SEKE '02*, pages 271–274, New York, NY, USA, 2002. ACM.

[8] T. Hoare. The Ideal of Program Correctness: *Third Computer Journal* Lecture. *Comput. J.*, 50(3):254–260, 2007.

[9] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, sixth edition, September 2004.

[10] Z. Manna and A. Pnueli. *Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.

[11] W. Sun, T. Shi, G. Argote-Garcia, Y. Deng, and X. He. Achieving a Better Middleware Design through Formal Modeling and Analysis. In *SEKE '06*, pages 463–468, 2006.

[12] The Eclipse Foundation. Graphical Editing Framework (GEF), 2007. `http://www.eclipse.org/gef/`.

[13] C. Zhang, S. M. Sadjadi, W. Sun, R. Rangaswami, and Y. Deng. A User-Centric Network Communication Broker for Multimedia Collaborative Computing. *COLCOM*, 0:28, 2006.

An Algorithm for Computing Loop Functions

Ali Mili, Shir Aharon, Chaitanya Nadkarni
College of Computing Science
New Jersey Institute of Technology, Newark NJ 07102
mili@cis.njit.edu, ShirAharon09@comcast.net, cgn4@njit.edu

Abstract

We consider a while loop on some space S and we are interested in deriving the function that this loop defines between its initial states and its final states (when it terminates). Such a capability is useful in a wide range of applications, including reverse engineering, software maintenance, program comprehension, and program verification. In the absence of a general theoretical solution to the problem of deriving the function of a loop, we explore engineering solutions. In this paper we discuss the design and preliminary implementation of a tool that derives or approximates the function of while loops written in C-like languages.

Keywords

Reverse engineering; software maintenance; program comprehension; while loops; program semantics; program correctness; refinement calculi; software tools.

1. Introduction

As software is used in increasingly critical applications, it is getting increasingly important to ensure its correctness, and to analyze/ understand its function. Simultaneously, as software grows increasingly large and complex, it is getting more and more difficult and costly to do so to an adequate level of confidence. Furthermore, recent software development paradigms (software reuse, product line engineering, COTS based software development, outsourcing, etc) are heavily dependent on third party software products, whose quality cannot be ascertained by process controls (process standards, process maturity levels, etc); this places the burden of quality assurance on analyzing the resulting product. The convergence of these three trends places a great premium on automated tools that allow us to analyze the function of software components and software systems to an arbitrary level of thoroughness and precision.

Deriving or approximating (characterizing) the function of a software system involves reasoning at many different levels of the software hierarchy, and modeling many aspects of interaction between the components of a complex system. At the lowest level, the source code level, one of the most challenging tasks is the derivation or the approximation of loop functions. In this paper, we present an algorithm that derives the function of a while loop from a static analysis of its source code; we also discuss and illustrate a current implementation of the algorithm, as well as venues for its future evolution. In the next section we showcase the current capability of our algorithm by means of a sample program, whose function we compute using our algorithm. Then, in section 3 we present the broad structure of our algorithm, and discuss its current status of development. In section 4 we briefly present the mathematical foundations of the algorithm, and use these to present the detailed structure of the algorithm, in section 5. In section 6 we assess the proposed algorithm, outline its future evolution in light of this assessment, and briefly discuss related work.

2. Brief Illustration

We consider the C++ program given below and we are interested to derive the function of its loop. This program handles integer variables, and also includes arrays, lists and (symbolic) function calls.

```
1.  #include <iostream>
2.  #include <cmath>
3.  #include <math.h>
4.  #include <list>
5.  using namespace std;
7.  const int a= , b= , c= , d= , e= ;
8.  const int N= ;
91  typedef list <int> listtype;
10  listtype l, m;  int q, qc;
11  int x, y, z, t, i, j, v, w, SA, Sn;
12  int A[N], B[N];
13
14  void loop ();
15  int f (int x);
16  int main ()
17     {loop();}
18  void loop  ()
19     { while (i != 0)
```

```
20          {y = y+b;  v = v+a*t;
21           w = w+e*y-b*e;
22           x = x+a;  t = t*d;
23           sA = sA + A[i];
24           sB = sB + B[j];
25           i = i-1;  j = j+1;
26           z = z+c*x-a*c;
27           m.push_back(l.front());
28           l.pop_front();
29           q = f(q);  qc = qc + q;}
30      }
31  int f (int x)
32      {return (//some function of
33       x);}
```

The function of this loop is given in figure 1 (where list concatenation is represented by a dot). It includes two terms: the trivial term where $i = 0$ and all variables are preserved; the non-trivial term where $i \neq 0$ and program variables are altered. This figure gives the final values (primed) of the program variables as a function of the initial values (unprimed).

For the sake of comparison, we submitted the same program to Daikon [5], which generates loop invariants by applying machine learning techniques to the execution trace. Because it operates on execution traces (rather than on source code), Daikon requires that we fix all the constants (a significant loss of generality, since then it makes a statement not about a broad family of programs, but rather about a single program). Daikon did find some of the clauses of the function given in Figure 1, duly specialized to the constant values. In fairness, we must recognize that, because Daikon operates on execution traces, it can handle any program structure, whereas we can only handle program structures for which we have made prior provisions.

3 Broad System Structure

To derive the function of a loop written in a given programming language, we proceed in three steps.

1. *Map the loop from its source programming language notation to a predefined language-independent internal notation*. The internal notation is defined in such a way as to support the divide and conquer approach that we advocate. We make it language independent so as to support a wide range of programming languages with minimal overhead.

2. *We analyze the loop written in the internal notation to derive equations between the initial (unprimed) variables and the final (primed) variables*. This step is the core of our algorithm. We analyze small parts of the loop at a time with a view to answering the question: What equations hold between the initial values and the final values of the loop.

3. *We submit the equations derived in the previous step to a system for solving symbolic equations*. We obtain the function of the loop by solving the equations in the primed variables, using the unprimed variables as parameters. For now we are using *Mathematica* (©Wolfram Research), but we are also exploring other systems as well.

The first step is currently carried out by hand, but can easily be automated using compiler generation technology. The third step is fairly trivial, since the equations generated by the second step are written directly in Mathematica notation. The second step is the focus of our subsequent discussion. It is automated by means of a C++ program, whose capability depends on storing pattern matching artifacts, as we discuss below.

4. Mathematical Foundations

Space restrictions preclude us from a detailed discussion of the mathematical background to the proposed work; the interested reader is referred to [11]. Suffice it to say, for the purposes of our discussion, that: we represent program specifications by relations and program functions by deterministic relations; specifications (represented by relations) are ordered by refinement, which we denote by $\sqsupseteq$; the refinement ordering is a partial ordering, and has lattice-like properties, where the join is represented by $\sqcup$; if a specification R refines two specifications R_1 and R_2 then it refines their join $R_1 \sqcup R_2$; the lattice of refinement has a universal lower bound but has no universal upper bound; maximal elements of the lattice are total deterministic relations.

4.1. Approximating a Loop Function

We consider a while loop of the form: `while t do B` on some space S and we let W be the function of this loop; we assume that this loop terminates for all initial states in S.Our stepwise approach to the derivation of the loop function is that we obtain this function by accumulating a sufficient number of (in)equations of the form $W \sqsupseteq T$, where T is some relation on S; we refer to T as a *lower bound* of W. By virtue of lattice properties of the refinement structure,if W refines T and T' then it refines their join. In practice, if we find a set of lower bounds $T_1, T_2, T_3, ...T_k$ to W, then we can infer:

$$W \sqsupseteq T_1 \sqcup T_2 \sqcup T_3 \sqcup ... \sqcup T_k.$$

By virtue of the structure of the refinement lattice, if the join of all the T_i is total and deterministic, then it is maximal in the refinement ordering, whence

$$W \sqsupseteq T_1 \sqcup T_2 \sqcup T_3 \sqcup ... \sqcup T_k \Leftrightarrow W = T_1 \sqcup T_2 \sqcup T_3 \sqcup ... \sqcup T_k.$$

In such cases, we have found the function of the loop. If, on the other hand, the join of all the lower bounds we have

$$\{ \begin{pmatrix} x & x' \\ y & y' \\ z & z' \\ v & v' \\ w & w' \\ t & t' \\ i & i' \\ j & j' \\ sA & sA' \\ sB & sB' \\ A & A' \\ B & B' \\ l & l' \\ m & m' \\ q & q' \\ qc & qc' \end{pmatrix} \mid \begin{array}{c} d \neq 1 \wedge abdei \neq 0 \wedge i' = 0 \wedge t' = d^i t \wedge v' = \frac{(atd^i + vd - at - v)}{(d-1)} \wedge \\ w' = \frac{bei^2 - bei + 2eyi + 2w}{2} \wedge x' = x + ai \wedge y' = y + bi \wedge z' = \frac{aci^2 - aci + ecxi + 2z}{2} \\ sA' = sA + \sum_{k=1}^{i} A[k] \wedge sB' = sB + \sum_{k=j}^{j+i-1} B[k] \wedge j' = j + i \wedge \\ q' = f^i(q) \wedge qc' = qc + \sum_{k=1}^{i} f^k(q) \wedge \\ i \leq length(l) \wedge l' = Rest^i(l) \wedge m'.Rest^i(l) = m.l \end{array} \} \cup$$

$$\{ \begin{pmatrix} x & x' \\ y & y' \\ z & z' \\ v & v' \\ w & w' \\ t & t' \\ i & i' \\ j & j' \\ sA & sA' \\ sB & sB' \\ A & A' \\ B & B' \\ l & l' \\ m & m' \\ q & q' \\ qc & qc' \end{pmatrix} \mid \begin{array}{c} i = 0 \wedge abd^2e \neq abde \wedge x' = x \wedge y' = y \wedge z' = z \wedge \\ t' = t \wedge v' = v \wedge w' = w \wedge i' = 0 j' = j \wedge sA' = sA \wedge sB' = sB \wedge \ \}. \\ m' = m \wedge l' = l \wedge q' = q \wedge qc' = qc \wedge A' = A \wedge B' = B \end{array}$$

Figure 1. Function of the Sample C++ Program

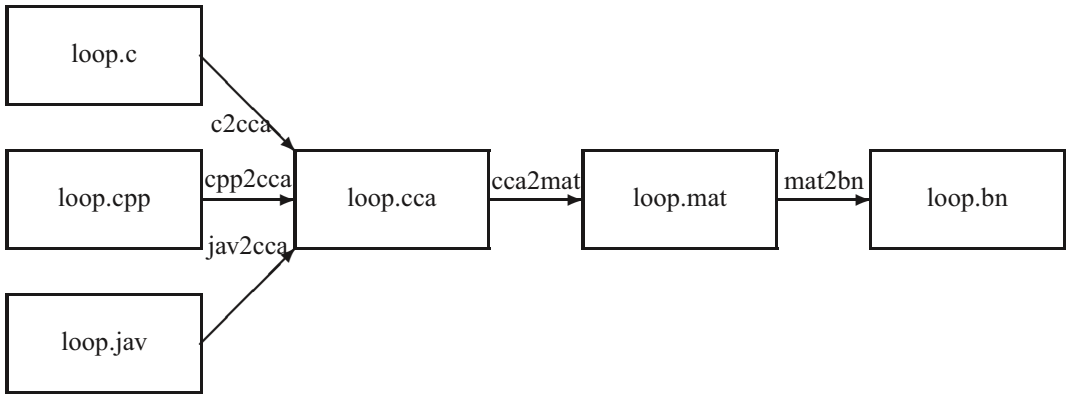


Figure 2. Broad Architecture of the Tool

found is not a total function, then we do not have the function of the loop, but we have an approximation of it.

The following results, which we present without proof (the interested reader is referred to [11]), are geared towards finding lower bounds of W.

Theorem 1 *We consider the while statement* `while t do B`*, where* $t \neq$ **false** . *Then*

$$T = I(t) \circ L \circ I(t) \circ [B] \circ I(\neg t) \cup I(\neg t)$$

is a lower bound for W.

Theorem 2 *If* R *is a reflexive transitive relation that is a superset of* $[B]$ *such that* $R \circ I(\neg t)$ *is total then* $T = R \circ I(\neg t)$ *is a lower bound of* W.

While theorem 1 gives an explicit expression of a lower bound, theorem 2 relies on us to derive relation R that satisfies some conditions; once R is found, this theorem prescribes how we infer a lower bound from it.

5. Detailed Algorithm

5.1. The Internal Representation

Because theorem 2 requires that we find a superset of the loop body, we must represent the loop body in a way that makes supersets visible. In typical programming languages, the loop body is represented as a sequence of statements, a structure which does not lend itself to finding supersets: in order to find the superset of a sequence, we must look at each term of the sequence. To obviate this difficulty, we propose to represent the loop body as an intersection instead of a sequence: indeed, if B is written as

$$B = B_1 \cap B_2 \cap B_3 \cap ... \cap B_n,$$

then a superset of B_1 is a superset of B, a superset of $B_1 \cap B_2$ is a superset of B, a superset of $B_1 \cap B_2 \cap B_3$ is a superset of B. The notation we have chosen to this effect is what is called *(Conditional) Concurrent Assignments*, or *CCA*'s for short. These represent variable assignments that are carried out concurrently, or in an arbitrary order.

5.2. Deriving Lower Bounds

Once the loop body is structured in CCA form, we can derive lower bounds by looking at one statement at a time, or two statements at a time, or three statements at a time, etc. To derive lower bounds of loop functions, we scan their loop body written in CCA form, match their statements or combinations of statements against pre-cataloged code patterns, and derive duly instantiated lower bounds in case of a match. We use the term *recognizer* to refer to the aggregate made up of variable declarations, code patterns, and corresponding lower bound; and we distinguish between *one-recognizers* that match one statement at a time, *two-recognizers* that match two statements at a time, *three-recognizers* that match three statements at a time. The current status of development of the extraction algorithm can be characterized by the following statements:

- All the machinery for recognizing code patterns and generating instantiated lower bounds is currently in place.
- We have a total of 28 recognizers, including ten 1-recognizers, fifteen 2-recognizers, and three 3-recognizers.

We can augment the scope of applicability of the algorithm by adding more recognizers, to handle new control structures and new data structures. Table 3 shows some sample recognizers that are currently implemented. For the sake of brevity, we do not show the term $\neg t(s')$ in the lower bounds, although it should be there, by virtue of theorem 2.

The question of how recognizers are derived is beyond the scope of this paper; suffice it to say that they are derived using the concept of *strongest invariant functions* introduced in [12], and that they are discussed in greater detail in [11].

5.3. Combining Lower Bounds

The join of all the lower bounds is itself a lower bound of the loop function. If this join is a total deterministic relation (a total function) then it is the function of the loop; else it is a lower bound of the function of the loop (i.e. it specifies some, but not all, of the functional properties of the loop). In practice, if Mathematica returns an expression for each primed state variable, and no restriction on the unprimed state variables, then we have found the function of the loop (because then the equations that represent the join of the lower bounds define a total deterministic relation).

5.4. Illustration

For the sake of illustration, we consider the loop presented in section 2 and we present in turn excerpts of the loop written in the CCA format, then excerpts of the Mathematica file produced by the recognizers.

loop.cca:

```
{
const int a; const int b; const int c;
const int d; const int e; const int N;
const function f;
array int A; array int B;
list l;  list m;
int q; int qc;
int x; int y; int z; int t; int i;
int j; int v; int w; int sA; int sB;
while !(i == 0)
  {v = v+a*t, z = z+c*x, w = w+e*y,
```

ID	State Space	Code Pattern	Lower Bound $T =$
1R1	x: int; const c: int >0	x=x+c	$\{(s,s')\|x \bmod c = x' \bmod c\}$
1R2	x: int	x=x+1	$\{(s,s')\|x \leq x'\}$
1R3	x: int	x=x-1	$\{(s,s')\|x \geq x'\}$
2R1:	x, y: int; const a, b: int	x = x+a, y=y+b	$\{(s,s')\|ay - bx = ay' - bx'\}$
2R2:	x, y: int; const a: int	x = x*a, y=y+x	$\{(s,s')\|y(1-a) + x = y'(1-a) + x'\}$
2R3:	x, y: int; const a, b: int	x = x+a, y=y*b	$\{(s,s')\|\frac{y}{b^{x/a}} = \frac{y'}{b^{x'/a}}\}$
2R4:	x, y: listType	y:=y.First(x), x:= Rest(x)	$\{(s,s')\|y.x = y'.x'\}$
2R5:	i: int;x: sometype	i:=i-1, x:=f(x)	$\{(s,s')\|f^i(x) = f^{i'}(x')\}$
3R1:	i: int; x,y: sometype	i:=i-1, x:=f(x), y:=y+x	$\{(s,s')\|y + \Sigma_{k=1}^{i} f^k(x) = y' + \Sigma_{k=1}^{i'} f^k(x')\}$
3R2	x: int; a[N]: int; i: int	i=i+1, x=x+a[i], a=a	$\{(s,s')\|a' = a \wedge x + \sum_{k=i}^{N} a[k] = x' + \sum_{k=i'}^{N} a'[k]\}$
3R2	x: int; a[N]: int; i: int	i=i-1,, x=x+a[i], a=a	$\{(s,s')\|a' = a \wedge x + \sum_{k=1}^{i} a[k] = x' + \sum_{k=1}^{i'} a'[k]\}$

Figure 3. 1-, 2-, and 3-Recognizers

```
   x = x+a, y = y+b, t = t*d,
   sA = sA+A[i], sB = sB+B[j],
   i = i-1, j = j+1, l = tail(l),
   m = m.head(l), q = f(q),
   qc = qc+q, A = A, B = B}
}
```

The algorithm produces 56 equations, of which we present the following excerpts:

loop.mat

```
1.  Reduce[ Reduce[ {
2.  Mod[x,Abs[a]]==Mod[xP,Abs[a]],
6.  i>=iP,
9.  A==AP,
11. v+a*t/(1-d)==vP+a*tP/(1-d),
12. z-c*x*(x-a)/(2*a)==
    zP-c*xP*(xP-a)/(2*a),
14. a*y-b*x==a*yP-b*xP,
16. t/d^(x/a)==tP/d^(xP/a),
17. a*i+1*x==a*iP+1*xP,
20. t/d^(y/b)==tP/d^(yP/b),
24. t/d^(j/1)==tP/d^(jP/1),
25. 1*i+1*j==1*iP+1*jP,
26. lP==Nest[Rest,l,i-iP],
27. i-Length[l]==iP-Length[lP],
28. Nest[f,q,i]==Nest[f,qP,iP],
30. j+Length[l]==jP+Length[lP],
31. Join[m,l]==Join[mP,lP],
32. sA+Sum[A[k], {k,1,i}] ==
    sAP+Sum[AP[k], {k,1,iP}],
34. qc+Sum[Nest[f,q,k],{k,1,i}]==
    qcP+Sum[Nest[f,qP,k],{k,1,iP}],
35. (iP==0),
41. {iP, jP, lP, mP, qP, qcP, sAP,
42.  sBP, tP, vP, wP, xP, yP, zP},
43. Backsubstitution->True]
```

Lines 1 and 43 are Mathematica instructions/ options. Lines 41 and 42 specify that we want the given equations resolved in these variables, which are the final values of the program variables. Lines 2 through 6 represent the application of 1-recognizers. Lines 9 to 31 represent the application of 2-recognizers. And lines 32 to 34 represent the application of the 3-recognizers. Line 35 represents the clause $\neg t(s')$ that we have factored out from all the lower bounds.

6. Assessment and Prospects

6.1. Related Work

Our work is related to three lines of research: research on deriving loop functions, with which it shares a common goal; research on deriving loop invariants, with which it shares common analytical methods; and research on program slicing, with which it shares common divide-and-conquer approaches. We discuss these in turn, below.

The closest work we have found to our effort, in terms of goal (generating loop functions) and means (using Mills-like functional/ relational logic) is work by Dunlop and Basili [4]. In this work, Dunlop and Basili discuss a syntactic method that derives the function of a loop by attempting to generalize from known formulas that capture the behaviors of the loop under special conditions.

Generally, the derivation of loop invariants is closely related to the derivation of loop functions since they both aim to discover the inductive argument that underlies the behavior of the loop. Many researchers in the theorem proving and the program verification communities have lent much attention to the goal of extracting loop invariants. In [5] Ernst et al. discuss a system for dynamic detection of likely invariants; this system, called Daikon, runs candidate programs and observes their behaviors at user-selected points, and reports properties that were true over the observed executions, using machine learning techniques. In [3], Denney and Fischer analyze generated code against safety properties, for the purpose of certifying the code. In [2], Colón et al. consider loop invariants of numeric programs as linear

expressions and derive the coefficients of the expressions by solving a set of linear equations; they extend this work to non linear expressions in [13]. In [9] Kovacs and Jebelean derive loop invariants by solving recurrence relations; they pose the loop invariants as solutions to recurrence relations, and derive closed forms of the solution using a theorem prover (Theorema) to support the process. In [1] Rodriguez Carbonnell et al. derive loop invariants by forward propagation and fixed point computation, with robust theorem proving support. In [10], we discuss the difference between traditional loop invariants (in the sense of Hoare's logic [7, 6]) and the loop invariants that we derive in this paper from invariant functions, which we call *reflexive transitive loop invariants*.

In [8] Hu et al present a technique for slicing while loops while attempting to minimize slice sizes. The technique is based on identifying the induction variable of the loop, and applying semantics-preserving transformations that represent the effect of the loop by an if-then-else statement. Our work differs from that of Hu et al in many ways, including: first, we do not need to identify an inductive variable; second, our lower bounds can be arbitrarily partial, as they are not driven by the syntactic structure of the loop (while slicing techniques slice the program, our divide-and-conquer techniques slices the program's function); third the relation of our lower bound to the function of the loop is well defined (refinement), as is the rule for composing lower bounds (join).

7. Conclusion

The goal of computing program functions, notably for iterative programs, is a difficult goal, but is nevertheless a worthwhile goal, given the advances that it affords us in terms of program comprehension, program analysis, reverse engineering, software maintenance, software inspection, etc. In this paper, we outline an algorithm for computing loop functions, and illustrate its behavior on a simple example. The current algorithm has all the necessary infrastructure to derive Mathematica equations; the capability of the algorithm evolves through the addition of new recognizers. In the short term, the bottleneck of this process is that we can only generate symbolic equations that Mathematica can resolve. Yet new application domains involve domain-specific knowledge, whose integration requires an inference capability; we are not sure yet whether Mathematica can fulfill this need. Another bottleneck, that may arise in the medium term as the number of recognizers grows, is the need to control redundancy; while we have many ideas on how to do this, they are all likely to significantly increase the complexity of the algorithm. An equally pressing need, of course, is the ability to deal with conditionals; we have a theorem (not presented in this paper, but alluded to) that supports this step, using relational identities. We fully expect such a solution to increase the complexity of the algorithm; in particular, it will involve a more intensive interaction between the recognizer-based matching and the symbolic equation manipulation of Mathematica.

On balance, we argue that the proposed approach is worthy of further investigation, as it takes an angle to the analysis of while loops that is fairly orthogonal to existing approaches, and is likely to complement their results and their insights.

References

[1] E. R. Carbonnell and D. Kapur. Program verification using automatic generation of invariants. In *Proceedings, International Conference on Theoretical Aspects of Computing '2004*, volume 3407, pages 325–340. Lecture Notes in Computer Science, Springer Verlag, 2004.

[2] M. A. Colon, S. Sankaranarayana, and H. B. Sipna. Linear invariant generation using non linear constraint solving. In *Proceedings, Computer Aided Verification, CAV 2003*, volume 2725 of *Lecture Notes in Computer Science*, pages 420–432. Springer Verlag, 2003.

[3] E. Denney and B. Fischer. A generic annotation inference algorithm for the safety certification of automatically generated code. In *Proceedings, the Fifth International Conference on Generative programming and Component Engineering*, Portland, Oregon, 2006.

[4] D. Dunlop and V. R. Basili. A heuristic for deriving loop functions. *IEEE Transactions on Software Engineering*, 10(3):275–285, May 1984.

[5] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 2006.

[6] D. Gries. *The Science of programming*. Springer Verlag, 1981.

[7] C. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576 – 583, Oct. 1969.

[8] L. Hu, M. Harman, R. Hierons, and D. Binkley. Loop squashing transformations for amorphous slicing. In *Proceedings, 11th Working Conference on Reverse Engineering*. IEEE Computer Society, 2004.

[9] T. J. L. Kovacs. An algorithm for automated generation of invariants for loops with conditionals. In D. Petcu, editor, *Proceedings of the Computer-Aided Verification on Information Systems Workshop (CAVIS05), 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC05)*, pages 16–19, Department of Computer Science, West University of Timisoara, Romania, 2005.

[10] A. Mili. Reflexive transitive loop invariants: A basis for computing loop functions. In *First International Workshop on Invariant Generation*, Hagenberg, Austria, June 2007.

[11] A. Mili, S. Aharon, M. Pleszkoch, and R. Linger. Towards the automated derivation of loop function. Technical report, NJIT, http://web.njit.edu/mili/fxloop.pdf, September 2007.

[12] A. Mili, J. Desharnais, and J. R. Gagne. Strongest invariant functions: Their use in the systematic analysis of while statements. *Acta Informatica*, April 1985.

[13] S. Sankaranarayana, H. B. Sipna, and Z. Manna. Non linear loop invariant generation using groebner bases. In *Proceedings, ACM SIGPLAN Principles of Programming Languages, POPL 2004*, pages 381–329, 2004.

Verifying Behavioral Correctness of Design Pattern Implementation

Tu Peng, Jing Dong, Yajing Zhao
Department of Computer Science
University of Texas at Dallas, Richardson, TX 75083, USA
{txp051000, jdong, yxz045100}@utdallas.edu

Abstract

Design pattern describes a recurring problem and its common solution, which usually is in abstract form. The application of design pattern requires coding the generic solution. It is necessary to assure the coding process correctly implements not only the structure but also the desired behavior of the design pattern. This problem is called implementation correctness in this paper. By providing the definition of partial order between sequence diagrams, we formally describe the implementation correctness. We verify the implementation correctness with model checking by using process algebra to specify the source code and temporal logic to specify the behavior of the pattern.

KEYWORDS: Design pattern, model checking, temporal logic, partial order, process algebra

1. Introduction

Design pattern [12] is a reusable software development strategy. While design pattern documents expert experience, the correctness of its implementation is critical to assure the quality of the software system. Each design pattern is usually described by its intent, motivation, structural and behavioral solutions, consequences, known uses, and etc. The structural and behavioral solutions are typically modeled by class and sequence diagrams, respectively. In this paper, we focus on analyzing the behavioral correctness of design pattern implementation by exploiting the partial order relationship between the sequence diagram of a general design pattern and that of its implementation. We identify this problem as *implementation correctness*: given a design pattern *X* and its implementation program *P*, is the program the correct implementation of *X*? For example, consider the source code shown in [21] which is adopted from [19]. This source code is claimed to be an implementation of the Observer pattern [12]. However, how could we justify this? More generally, how do we know whether the implementation of a design pattern satisfies its behavior? To know whether the behavior of a pattern is implemented correctly is critical in assuring software reliability.

Formal methods have been widely used in verification because of two advantages. First, formal verification renders rigorous results. Second, formal verification can be facilitated by tool support, e.g. a model checker, thus less human efforts and human errors are involved. However, there are three obstacles to formally verify design pattern implementation. First, design pattern is more like guidance rather than any concrete algorithm; hence it is not easy to formally describe what rules to be verified. Second, design pattern implementation may be different from the original description, in terms of class names, method names, class numbers, method numbers, and so on. This makes it difficult to apply a single algorithm to verify the correctness of different implementations. Third, regarding to the correctness of behavioral characteristics of patterns, software program which has a large number of states can cause significant runtime delay and even state explosion when using model checker.

Our approach to tackle these obstacles is based on the following ideas. First, since the behavior of design patterns can be normally modeled by sequence diagrams, we abstract the verification rules from sequence diagrams and use temporal logics, e.g., CTL [10] and its extensions, to specify them. Second, we introduce anonymous specification, so that the implementation with different class/method names can be checked against the original design pattern. Third, we use CCS [15] to specify the sequence diagram recovered from the program, instead of the program itself. This makes the resulted system less complex by involving fewer states.

There have been many researches on the formal specification and verification of design pattern. Some approaches [1][3][11][14][17] focused on the formal specification of design pattern. Other approaches [4][5][6][8] discussed the verification of design pattern applications, supported by model checker. Previous works mainly focus on verifying the properties of design patterns at the design level, e.g., the liveness and safety properties of design pattern [6], the correctness of design pattern composition [4], or the security design patterns [8]. Little effort on verifying design pattern implementation has made, which is very important to assure the reliability of software system [13].

In the next section, we introduce the partial order between sequence diagrams, from which we can formally define the implementation correctness problem. In Section 3, we discuss in detail how to verify the implementation against the design pattern. Related theorems and algorithms are also presented. In Section 4, we present a case study to demonstrate the algorithms for implementation correctness problem, and provide other application which can be reduced to the same problem. We conclude this paper in Section 5.

2. Partial Order of Sequence Diagrams

Sequence diagram reflects the order of the method invocations in an object, and the interaction between different objects in a software system. There are three advantages of using sequence diagram to capture the behavior of a design pattern or a program. First, the methods, the order of their occurrence and their interactions are rigorously described, which make it easy for formalizing. Second, there exist tools which are able to recover sequence diagrams from source code. This can release human from analyzing source code directly. Third, sequence diagram is an abstraction of the behavior of objects, which involves less variables and states; hence the system derived from a sequence diagram is more suitable for model checking [2]. Figure 1 displays the sequence diagram that models the behavior of the general Observer pattern. Figure 2 shows the sequence diagram of the source code in [21] which is an implementation of the Observer pattern. Sequence diagrams can be automatically generated from the source code by tools, e.g., Together [20].

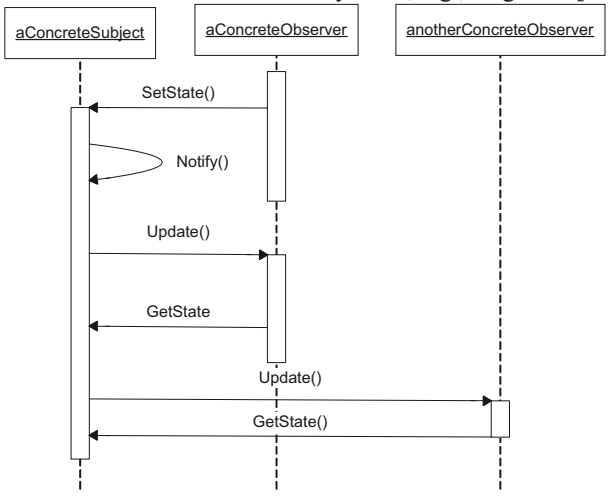


Figure 1 sequence diagram of observer pattern

Our goal can now be reduced to studying the relationship between the sequence diagram in Figure 1 and that in Figure 2. If the sequence diagram of the implementation satisfies that of the design pattern, it is reasonable to believe that the implementation is correct. This relationship between two sequence diagrams is defined in partial order in this paper.

There have been several researches on discovering design pattern from source code [7][9][16], so that the original design decisions of the source code can be recovered. The correctness of the recovery can actually be reduced to checking the partial order relationship between the sequence diagram of the recovered pattern and that of the source code. We will discuss how the partial order is formally defined in the rest of this section, how the partial order can be automatically checked in Section 3, and how the partial order can be applied to solve practical problems in Section 4.

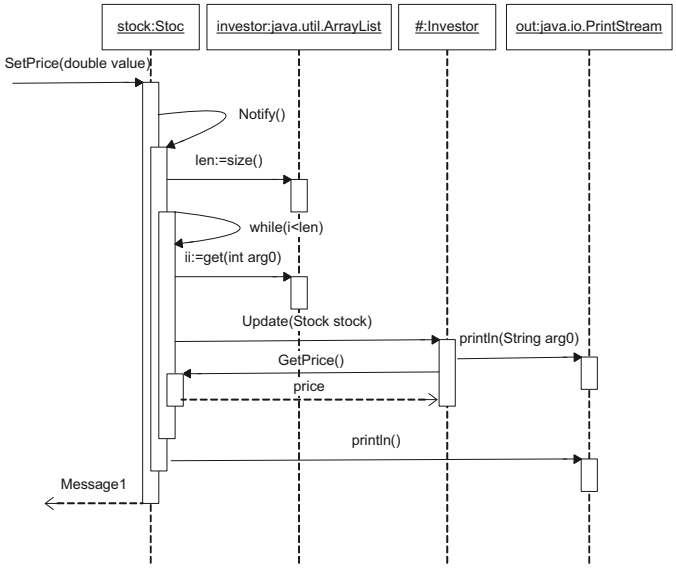


Figure 2 Sequence diagram of the implementation

In our approach, we found the order of the actions in a sequence diagram is the key factor to decide the relationship of two sequence diagrams. Hence we first define a convenient notation to denote the time order between actions.

Definition 1 Suppose a and b are two actions. We use $a < b$ to denote a occurs before b, and $a > b$ to denote a occurs after b.

As the sequence diagrams may come from different sources, e.g., from a design model by a designer or recovered from a piece of source code, the actions usually are named differently although they may define the same sequence of actions. Hence we are interested in the order of two different sets of actions, rather than the action names, from two different sequence diagrams. We define the order of actions as follows.

Definition 2 Suppose A is a set of actions and f is a one-one function with domain and range over A. Suppose t is a set of actions, which are mapped into another set of actions $f(t)$ by function f. We say actions in t and actions in $f(t)$ have the same order, denoted by $t \approx f(t)$ when the following formula applies,

$$\forall t_1, t_2 \in t:$$

$$(t_1 > t_2 \rightarrow f(t_1) > f(t_2)) \wedge (t_1 < t_2 \rightarrow f(t_1) < f(t_2)).$$

This definition specifies that the renaming function f reserves the order of the actions in t.

Lemma 1 If $t \approx f(t)$ and $s \subseteq t$, then $s \approx f(s)$.

The proof is obvious.

Definition 3 Suppose D_1 and D_2 are two sequence diagrams and t is a set of actions. We say D_1 satisfies D_2 with

respect to *t*, expressed by the formula $D_1 \succ_t D_2$, if and only if the following conditions are observed:

1) D_2 contains all the actions of *t*, and D_1 contains a set of actions t_1, which can be mapped one by one onto *t*. That is, there exists a one-one function *f* from *t* to t_1.
2) $t \approx f(t)$.

This definition states that one sequence diagram satisfies another sequence diagram.

Theorem 1 $\succ_t$ is a partial order.

Proof: We need to prove the transitive, asymmetric, and reflexive properties of $\succ_t$.

Transitive: suppose there are three sequence diagrams, D_0, D_1 and D_2, with $D_1 \succ_t D_2$ and $D_0 \succ_{t_1} D_1$. Since $D_1 \succ_t D_2$, there is a one-one function f that maps the actions from t to t_1. Since $D_0 \succ_{t_1} D_1$, there is a one-one function f_1 that maps the actions from t_1 to t_2. Then it is clear that $f_1 \circ f$ is the one-one function from t to t_2. On the other hand, for any two actions *a* and *b* from *t*, if *a* occurs before *b* in D_2, then *f*(*a*) occurs before *f*(*b*) in D_1 because $D_1 \succ_t D_2$. Then $f_1(f(a))$ occurs before $f_1(f(b))$ because $D_0 \succ_{t_1} D_1$. Hence $D_0 \succ_t D_2$.

Asymmetric: it is clear that $D_1 \succ_t D_2$ and $D_2 \succ_t D_1$ cannot be true at the same time. Otherwise, *t* may contains actions not in D_1.

Reflexive: it is obvious that $D_1 \succ_t D_1$.

Hence we complete the proof that $\succ_t$ is a partial order.

Informally, partial order describes to what extend two sequence diagrams are similar to each other. If a partial order exists between two sequence diagrams, one could say the behavior of one sequence diagram is captured or included in another sequence diagram.

Definition 4 Suppose $t_1, t_2, \cdots, t_n$ are *n* sets of actions, we write $D_1 \succ_{t_1,\cdots,t_n} D_2$ to mean that $D_1 \succ_{t_1} D_2$, …, $D_1 \succ_{t_n} D_2$ hold.

3. Verify Design Pattern Implementation

In this section, we will discuss how to verify if the implementation of a given design pattern is correct. We start by formally defining the implementation correctness.

Definition 5 Given a design pattern *X*, whose sequence diagram is D_X. Given a program *P*, whose sequence diagram is D_P. Given *n* sets of actions $t_1, t_2, \cdots, t_n$ which occur in D_X. Then we say *P* is a correct implementation of *X* with respect to $t_1, t_2, \cdots, t_n$, if and only if $D_P \succ_{t_1,\cdots,t_n} D_X$.

This definition formally specifies the implementation correctness problem. That is, by comparing the order of the actions in $t_1, t_2, \cdots, t_n$ (from the design pattern) and their correspondent actions (from the implementation), one could know whether the design pattern is correctly implemented. We will then propose an approach to verify whether $D_P \succ_{t_1,\cdots,t_n} D_X$.

Given design pattern *X* and program *P*. The following is the outline of our approach.

1) For a given design pattern *X*, D_X is known. That is, the behaviour of a given pattern is known and is used as the standard to be verified against.
2) For *X*'s implementation *P*, D_P can be obtained automatically by using a software tool which can recover the sequence diagram from a program.
3) For all actions in $t_1, t_2, \cdots, t_n$, we use temporal logic to specify their existence and the order of their occurrence. This specification consists of a set of temporal logic properties, which is denoted by $PROP_X$.
4) Formally specify D_P, using CCS that is a process calculus, and obtain the formal expression of D_P denoted by CCS_P, which is a set of processes.
5) Verify if $CCS_P \models PROP_X$ is true, which will be defined in Definition 6.

We use the CCS (calculus for communicating system) [15] as the specification language. CCS is a process algebra which describes labelled transition system where several subsystems communicate with each other. The syntax of CCS is shown as follows

$$A ::= a.A \mid a'.A \mid A \mid A \mid A + A \mid A \setminus L \mid A[f].$$

where *A* is a CCS process. *a* is an incoming message and *a'* is an outgoing message. *a.A* means after accepting message *a*, process *A* happens, where "." is sequential operator; *A*|*A* means process *A* and *A* are concurrent, where "|" is parallel composition operator; *A* + *A* means either one of the two processes can happen, where "+" is summarization operator; *A**L* means all the messages of *A* which are included in set *L* are restricted as internal message, and "\" is restriction operator; *A*[*f*] means that the messages of *A* are renamed by the rule provided by *f*, and [] is referred as relabel operator.

Definition 6 $CCS_P \models PROP_X$ holds when every property in $PROP_X$ is satisfied by CCS_P. More specifically, for every property p_i in $PROP_X$, there exists a process in CCS_P which satisfies p_i.

The following definition provides a way to compute $PROP_X$. We will specify the order of the actions in set $t_1, t_2, \cdots, t_n$, in terms of temporal logic. These temporal logic specifications are usually called properties. $PROP_X$ is actually a set of these properties.

Definition 7 $PROP_X$ is a set of temporal logic properties, which specify the existence and order of the actions in t_i. Namely, $PROP_X = \{1 \le i \le n \mid p_i\}$. For each action a in t_i, suppose f is the function defined on t_i. Suppose $m \subseteq t_i$ such that $f(m) = m$:

1) Let p_i specify the existence of all actions in t_i.
2) Let p_i specify the order of occurrence of all actions in m by referring their names.
3) Let p_i specify the order of occurrence of all actions in $t_i - m$ without referring their names.

There are a few points about this definition that need to be mentioned. First, it is not fully automatable and human efforts are needed in every step. Second, the actions in $t_i - m$ have different names in the implementation from those in the design pattern, which need to be specified anonymously, so that the name difference between design pattern and its implementation can be tolerated in model checking.

In the rest of this section, we present two algorithms for our approach. Algorithm 1 provides the steps to compute CCS_P, which is an abstract model of program P. CCS_P is a set of processes specifying the actions of objects in D_P. Algorithm 2 provides steps to check $CCS_P \models PROP_X$,

Algorithm 1

For each class C appearing in D_P, the specification of C's actions, $spec_C$, is obtained by the following steps:

1) For each activation bar in a sequence diagram, specify its actions with sequential order. For all the outgoing actions, specify them with a prime symbol before their names. For all the other messages, just specify them with their names.
2) Connect the specification of each activation bar with summarization operation.
3) Add a recursion and we complete the specification of the actions performed by C.

$CCS_P = \{C \ from \ D_P \mid spec_C\}$

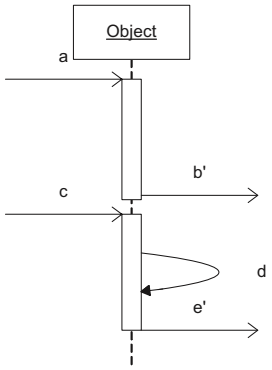


Figure 3 A Simple Sequence Diagram

Let us use an example to illustrate how Algorithm 1 is used to specify a sequence diagram in CCS. Figure 3 shows a simple sequence diagram. Its CCS process is {a.b'+c.d.e'}, where a.b' is obtained through the first activation bar and c.d.e' is obtained from the second one.

Algorithm 2

For every property r in $PROP_t$, if there exist a process s in CCS_P, such that s satisfies r, then r is checked by model checker CWB-NC [18].

Theorem 2 Specification of D_P is automatable.

Proof: This comes naturally from the existence of the Algorithm 1 to specify D_P.

Theorem 3 $CCS_P \models PROP_X$ holds in Definition 7 if and only if $D_P \succ_{t_1,\cdots,t_n} D_X$ holds.

Proof: On one hand, if $D_P \succ_{t_1,\cdots,t_n} D_X$ holds, then $D_P \succ_{t_i} D_X$ holds for $1 \le i \le n$. That is, $t_i \approx f(t_i)$. Let $p_i \in PROP_X$ be the temporal logic property specifying t_i. Since $t_i \approx f(t_i)$, which means the actions in t_i and their correspondence in $f(t_i)$ happen in the same order. It follows that the process containing actions in $f(t_i)$ must satisfy p_i. Hence $CCS_P \models PROP_X$ holds.

On the other hand, if $CCS_P \models PROP_X$ holds, then all the properties in $PROP_X$ are satisfied by P. Hence the actions specified by the properties must happen in the same order as their correspondence in P. This would mean that $D_P \succ_{t_i} D_X$ holds for $1 \le i \le n$. Thus we complete the proof.

This theorem demonstrates that it is reasonable for us to use model checking to check $CCS_P \models PROP_X$, so as to determine whether $D_P \succ_{t_1,\cdots,t_n} D_X$ holds.

4. Case Study

In this section, we apply our approach presented in Section 3 to study the case in Section 2, that is, whether the source code is the correct implementation of the Observer pattern. As described in [12], the behaviour of the Observer pattern can be specified by the sequence diagram shown in Figure 1. The basic property of the Observer pattern can be summarized: 1) whenever the setstate in Subject is invoked, 2) the notify action must be performed, 3) the update in every Observer must be invoked, followed by the getstate. Hence the problem is to verify whether $D_P \succ_t D_{observer}$, with t={notify,update,setstate,getstate}, where we assume the action names are case insensitive. In particular, we specify the following properties of the Observer pattern.

- The first property states that there must be an action that happens before "notify". This action can be "set-

state" or named differently. Since "setstate" can have different name in real implementation, this consideration is essential.

```
prop subject_order1=
(E F{setstate}-> (E F{notify}))\/ (E F{-
setstate}-> (E F{notify}))
```

- The second property states that actions "notify" and "update" must exist in the Subject. Symbol "'" is applied before update to denote this message is outgoing.

```
prop subject_exist=
E F{notify}/\ E F{'update}
```

- The third property states that action "update" must happen after the "notify" action.

```
prop subject_order2=
E F{notify}-> (E F {'update})
```

- The fourth property states that there must be an action happens after the "update" action. Usually this is "getstate", but can have different name.

```
prop subject_order3=
(E F{'update}-> (E F{getstate}))\/ (E
F{'update}-> (E F{-getstate}))
```

- The final property of process subject is the conjunction of the above properties.

```
prop subject=
subject_order1 /\ subject_order2 /\
subject_order3 /\ subject_exist
```

All these properties are specified in GCTL, which is an extension of traditional temporal logic CTL* [10] that is tailored for reasoning about systems whose transitions are labelled by actions. The syntax of GCTL is

$$S ::= p \mid \neg p \mid S \wedge S \mid S \vee S \mid \mathrm{A}\, p \mid \mathrm{E}\, p \mid \mathrm{G}\, p \mid \mathrm{F}\, p$$

$$P ::= \theta \mid \neg\theta \mid S \mid P \wedge P \mid P \vee P \mid \mathrm{X}\, P \mid P \bigcup P \mid P\, \mathrm{R}\, P$$

where S is state formula, P is path formula. p is atomic proposition, and θ is atomic action proposition. A is a universal quantifier which means the formula after A is true in every state starts from the current state. E is an existential quantifier which means that there exists a state following the current state, where the formula after E is true. G is a path universal quantifier which means the formula is true along all the states in the path from the current state. F is a path existential quantifier which means the formula is true in some state in the path from the current state. G, F are always used together with A and E. More detailed specification of the semantics of GCTL is in [18].

A practical implementation of the Observer Pattern is shown in [21], whose sequence diagram is recovered from its source code in Figure 2. Note that this discovery is automatically done by Together [20].

Then, we need to specify the behaviour of the Observer pattern formally in CCS as follows

```
proc observer=
(stock|investors|investor|printer)\{
setprice,get,update,getprice,println}

proc stock=setprice.notify.'size.'get
.'update.stock1
proc stock1=getprice.'println.stock

proc investors=
size.investors1+get.investor1
proc investors1=investors

proc investor=update.'println.investor1
proc investor1='getprice.investor

proc printer=println.printer1
proc printer1=println.printer
```

Once we obtain the CCS specification of the source code, we could simply verify it against the properties previously defined, with a model checker [18].

We save the properties of the Observer pattern, $PROP_t$, in the file "observer.gctl". We save the system specification of the sequence diagram of the program in the file "observer.ccs". Then we can model-check it with the following commands and results:

```
cwb-nc> chk -L gctl stock subject
Generating ABTA from GCTL* formula...done
Initial ABTA has 56 states.
Simplifying ABTA:
Minimizing sets of accepting states...done
Performing constant propagation...done
Joining operations...done
Shrinking automaton...done
Computing bisimulation...
Done computing bisimulation.
Simplification completed.
Simplified ABTA has 30 states.
Starting ABTA model checker.
Model checking completed.
Expanded state-space 29 times.
Stored 0 dependencies.
TRUE, the agent satisfies the formula.
Execution time (user,system,gc,real):
(0.047,0.000,0.000,0.047)
```

Comparing to a conventional approach of model checking, that is, to specify the system from source code, our approach is more efficient and fast, because the system model is concisely built and thus involves far less states than the model built from source code directly. Hence our approach reduces the possibility of state explosion and increases model checking speed. As shown in the runtime scripts above, our system model contains as few as 56 states in total and is simplified to only 30 states, due to the specification from sequence diagram instead of the source code. The running time is only 0.047 seconds.

5. Conclusions and Future Work

Design patterns have been widely adopted in software industry to reuse expert design experience. Use of design patterns generally involves implementing them in different forms. So far, there has been lack of methods to ensure the correctness of the design pattern implementation.

In this paper, we provide a method to formally verify design pattern implementation. We formally define the implementation correctness problem in terms of a partial order of two sequence diagrams. We obtain the system specification (CCS_P) by recovering the sequence diagram from the source code, and specifying it with CCS. We generate the properties ($PROP_X$) of the design pattern by using temporal logic to specify its behaviour, which is usually modelled by a sequence diagram (D_X). We verify CCS_P against $PROP_X$ with CWB-NC model checker. Our approach provides a formal standard and concrete method to solve the implementation correctness assurance problem. Moreover, since CCS_P is built concisely, it is easy and fast to use model checking in our approach.

Future work includes verifying the structural correctness of design pattern implementation and exploring other methods of verification. One possible way of verifying the structural correctness is to exploit the current work on design pattern recovery, which is able to extract structural information such as classes, methods, and their associations from source codes and compare such information with general design pattern to determine whether the system correctly implement the pattern from structural point of view. One alternative way of verifying the behavioral correctness is to specify design pattern and its implementation as two systems respectively, and check the behavioral relationship of those systems. Since specifying design pattern directly is a better way to capture design pattern behavior than specifying the summarized properties of the pattern, we could expect more accurate result in this approach.

There are other issues that can be addressed by the implementation correctness. For example, to recover design pattern from source code is an important task in reverse engineering [7][9][16]. However, there lacks research on how to formally specify and verify the correctness of design pattern recovery results. This problem may be addressed as the implementation correctness problem because program *P* correctly implements design pattern *X*, if and only if design pattern *X* is correctly recovered from program *P*.

References

[1] Paulo Alencar, Donald Cowan, and Carlos Lucena. A Formal Approach to Architectural Design Patterns. *Proceedings of the Third International Symposium of Formal Methods Europe (FME)*, pages 576–594, 1996.

[2] E.M Clarke, E.A. Emerson, A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS),* 8(2), April 1986.

[3] S. Chinnasamy, R. R. Raje, and Z. Liu. Specification of design patterns: An analysis. *Proceedings of the 7th International Conference on Advanced Computing and Communications*, pages 300–304, 1999.

[4] Jing Dong, Paulo Alencar, and Donald Cowan, Ensuring Structure and Behavior Correctness in Design Composition, *Proceedings of the 7th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS),* pp279-287, Edinburgh UK, 2000.

[5] Jing Dong, Paulo Alencar, and Donald Cowan. A Behavioral Analysis and Verification Approach to Pattern-Based Design Composition. *International Journal of Software and Systems Modeling, Springer Verlag*, 3(4):262–272, December 2004.

[6] Jing Dong, Paulo Alencar, and Donald Cowan, Automating the Analysis of Design Component Contracts, *Software - Practice and Experience,* Wiley, 36(1), pp. 27-71, Jan. 2006.

[7] Jing Dong, Dushyant S. Lad and Yajing Zhao, DP-Miner: Design Pattern Discovery Using Matrix, the Proceedings of the Fourteenth Annual IEEE International Conference on Engineering of Computer Based Systems (ECBS), pages 371-380, Arizona, USA, March 2007.

[8] Jing Dong, Tu Peng, Yajing Zhao, Model Checking Security Pattern Compositions, the Proceedings of the Seventh International Conference on Quality Software (QSIC), pages 80-89, USA, October 2007, IEEE CS Press.

[9] Jing Dong, Yajing Zhao, and Tu Peng, Architecture and Design Pattern Discovery Techniques – A Review, Proceedings of International Conference on Software Engineering Research and Practice (SERP), pages 621-627, USA, June 2007.

[10] E.A. Emersion and J.Y. Halpern. 'Sometime' and 'not never' revisited: On branching versus linear time temporal logic. *Journal of the Association for Computing Machinery*, 33(1):151-178, January 1986.

[11] A.H. Eden and Y. Hirshfeld. Principles in formal specification of object-oriented architectures. *Proceedings of the 11th CASCON, Toronto, Canada*, November 2001.

[12] Gamma E., Helm R., Johnson R. and Vlissides J. (1995). Design Patterns-Elements of Reusable Object-Ori*ented Software.* Reading, Massachusetts: Addison- Wesley.

[13] Peter C. Mehlitz, John Penix. Design for verification using design pattern to build reliable system. *Proceeding of 6th workshop on component-based software engineering, 2003*

[14] Tommi Mikkonen. Formalizing Design Pattern. *Proceedings of the 20th International Conference on Software Engineering*, pages 115–124, 1998.

[15] Robin Milner. Communication and Concurrency. Prentice Hall, 1989.

[16] Nija Shi and Ron Olsson. Reverse Engineering of Design Patterns from Java Source Code. *The International Conference on Automated Software Engineering*, Japan, Sept. 2006.

[17] Neelam Soundarajan and Jason O. Hallstrom. Responsibilities and Rewards: Specifying Design Patterns. *Proceedings of the 26th International Conference on Software Engineering*, pages 666–675, May 2004.

[18] CWB-NC User's Manual. http://www.cs.sunysb.edu/~cwb/

[19] http://www.dofactory.com/Patterns/Patterns.aspx

[20] http://www.borland.com/us/products/together/index.html

[21] http://www.utdallas.edu/~jdong/papers/Observer.java

Automated Multiperspective Requirements Traceability Using Ontology Matching Technique

Namfon Assawamekin[*], Thanwadee Sunetnanta[†], Charnyote Pluempitiwiriyawej[‡]
Department of Computer Science, Faculty of Science, Mahidol University
Bangkok 10400, THAILAND
Phone: +66(0) 2354-4333, Fax: +66(0) 2354-7333
[*]g4536828@student.mahidol.ac.th, [†]cctth@mahidol.ac.th, [‡]cccpt@mahidol.ac.th

Abstract

Large-scaled software development inevitably involves a group of stakeholders, each of which may express their requirements differently in their own terminology and representation depending on their perspectives or perceptions of their problems. However, those stakeholders will need to interoperate or collaborate by tracing, verifying and merging their requirements in order to achieve a common goal of their development. In this situation, ontology can play an essential role in communication among diverse stakeholders in the course of an integrating system.

This paper presents an alternative multiperspective requirements traceability (MPRT) framework to automatically generate traceability relationships of multiperspective requirements artifacts. Requirements ontology is designed and constructed as a knowledge management mechanism to represent multiperspective requirements artifacts in a common way, which intervene mutual "understanding" among various stakeholders. Ontology matching takes two ontologies and produces correspondences (i.e., equivalence, more general, less general, mismatch and overlapping) between the concepts of ontologies that correspond semantically to each other. As a result, the traceability relationships can be automatically generated when a match is found in the ontologies.

Keywords: *Interoperability, Multiperspective software development, Ontology, Requirements traceability*

1. Introduction

Nowadays, the development of most large and complex software systems inevitably involves many people or stakeholders. Different stakeholders may deal with different pieces of software requirements depending on their perspectives or perception of their problems. Each of the stakeholders may define his/her requirements in his/her own point of view using different terminologies. Such requirements are termed *multiperspective requirements artifacts*. A variety of stakeholders need to interoperate, collaborate or trace requirements among each other in order to achieve a common goal of their development. Consequently, the overlapping characteristic of multiperspective requirements artifacts makes it difficult to trace and verify the requirements.

As part of our attempt in resolving traceability problems of multiperspective requirements artifacts, we have investigated existing requirements traceability approaches and tools to manage software requirements and architecture. However, the proposed approaches for requirements traceability still encounter the following problems.

Firstly, the majority of existing commercial requirements traceability tools [1] utilize straightforward traceability links that must be manually defined by the users at coarse-grained level during software development process. Defining large number of traceability relationships manually can be a tedious, error-prone, labor-intensive and time-consuming task. The cost of using such tools for a large software development team is high on account of the licensing expense for each user. Additionally, the characteristic of these tools is the non-uniformity in dealing with the heterogeneity problems found in multiperspective requirements artifacts.

Secondly, automated solutions to requirements traceability problems face a difficult challenge due to the need to handle the overlapping among multiperspective requirements artifacts generated during software development process. For example, PROART [2] supports the automatic generation of traceability relationships among multiperspective requirements. Nevertheless, this approach forces the users to use the same set of vocabularies and their definitions stored in a repository. Alternatively, even though Huang [3] and Sherba [4] propose event-based

[*] The first author is also a lecturer at School of Science, University of the Thai Chamber of Commerce.

and open hypermedia approach respectively to automatically generate traceability links but they do not tackle the overlapping among multiple views of requirements.

Lastly, the granularity of requirements traceability is typically too coarse-grained level. For example, the works in RQML [5], XmlTRAM+ [6] and ROM & IREQ [7] aim to use XML as a mechanism for managing requirements artifacts or as the representation of requirements. However, RQML and XmlTRAM+ allow the users to generate traceability relationships manually at coarse-grained level, which are a very difficult and time-consuming task. Even if ROM and IREQ can automatically generate traceability relationships at fine-grained level, it is based on explicitly predefined rules.

To resolve the aforementioned problems, our proposed approach applies *requirements ontology* as a knowledge management mechanism in order to define an explicit account of a shared understanding among diverse stakeholders for knowledge interchange purposes. This work endeavors at handling *ontology interoperability* that does not force various stakeholders to express their requirements with the same or shared set of vocabulary, but supports multiple ontologies which represent different perspectives of stakeholders towards the same domain. *Ontology matching* is used to match the concepts between different and independent ontologies that correspond semantically to each other. The traceability relationships can be automatically generated when a match is found in the requirements ontologies.

The rest of this paper is organized as follows. Section 2 provides an overview of existing schema and ontology matching approaches, which are applied to our work for interoperability and traceability purposes. In section 3, a multiperspective requirements traceability (MPRT) framework is proposed. In section 4, we illustrate an example of how to automatically construct requirements ontologies from multiperspective requirements artifacts and produce their traceability relationships. Finally, we conclude the paper with the discussion of our ongoing work in section 5.

2. An Overview of Existing Schema and Ontology Matching Approaches

The purpose of this section is to survey on existing schema and ontology matching approaches in [8] and [9] which have emerged during the last decade. These approaches are classified into schema-based and instance-based systems. The main focus of this work is on schema-based matching (i.e., DIKE [10], TranScm [11], SKAT [12], ARTEMIS [13], Cupid [14], COMA [15], SF [16], CtxMatch [17], S-Match [18], COMA++ [19], DCM [20] and H-Match [21]). The instance-based and mixed systems are excluded from this consideration. We specifically apply *S-Match* [18] approach to match the concepts between requirements ontologies based upon the following observations.

Firstly, various approaches take as input a pair of schemas or ontologies, including *S-Match*, while only a small number of approaches take as input multiple schemas (e.g., DCM [20]).

Secondly, a large number of approaches handle graphs. Some examples include Cupid [14], COMA [15], SF [16], CtxMatch [17], COMA++ [19], H-Match [21] and *S-Match* [18].

Thirdly, most of existing approaches under consideration deal with particular schema or ontology types, such as relational, XML, RDF and OWL. Only a small number of approaches aim at being generic (i.e., handle multiple types of schemas or ontologies). Some examples include Cupid [14], COMA [15], SF [16], COMA++ [19] and *S-Match* [18].

Finally, most of existing approaches focus on computing similarity measures in [0, 1] range. Only little approaches compute semantic relations (i.e., equivalence, subsumption, mismatch, overlapping) between the concepts. Some examples of the latter include CtxMatch [17] and *S-Match* [18]. The semantic relations can be applied to our work for automatically generating traceability relationships.

3. Our Approach

The crucial goal of this work is to automatically generate traceability relationships of multiperspective requirements artifacts at fine-grained level, which can be applied to any software requirements domain. We propose an alternative multiperspective requirements traceability (MPRT) framework as depicted in Figure 1 in order to reach the goal.

Figure 1 illustrates our MPRT framework. The main modules in the framework can be summarized as follows:

1. *Requirements analyzer* obtains a set of requirements artifacts represented in terms of natural language or plain English text and uses the Stanford parser [22] to syntactically and semantically analyze the requirements artifacts.
2. *Requirements elements generator* (*REG*) generates the rules to extract requirements elements.
3. *Base ontology constructor* (*BOC*) uses IEEE Std 830-1998 [23] and ESA PSS-05-03 [24] to classify requirements types of requirements artifacts in the domain of software requirements.

4. *Requirements ontology constructor* (*ROC*) attaches requirements elements into the base ontology to automatically construct requirements ontology of each stakeholder as a common representation for knowledge interchange purposes.
5. *Ontology matcher* applies ontology matching technique in order to automatically generate traceability relationships between two requirements ontologies.

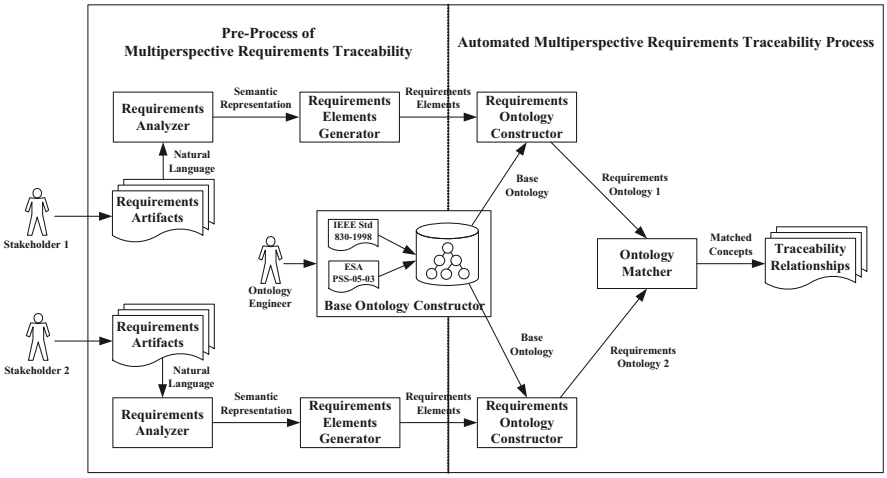


Figure 1. A Multiperspective Requirements Traceability (MPRT) Framework

3.1. Automatic Generation of Requirements Elements

Our REG module uses NLP techniques and a rule-based approach to extract requirements elements. We extract *objects* and *relationships* by using noun-verb analysis defined in CM-Builder [25] approach. We also use CM-Builder to generate the relationship between the objects (i.e., *aggregation*, *attribute*, *generalization/specialization* and *association*). However, CM-Builder approach only concerns the analysis of requirements text (static view) with the goal of building UML class models; it cannot recognize behavioral view of requirements artifacts for our work. As a consequence, we construct the transformation rules in SWI-Prolog [26] to extract the remaining requirements elements. The underline represents the grammatical relations [27] generated from the Stanford parser.

1. If an *object* participates in some prepositions (e.g. by) with a verb in the requirements artifacts, then the verb are taken as a *process* and the *object* as an *input* of the *process*.
2. If an *object* acts as a subject of a *process* in the requirements artifacts, then the *object* is taken as an *actor* of the *process*.
3. If an *object* acts as an object of a *process* in the requirements artifacts, then it is taken as the *output* of the *process*.
4. Two *objects* are the same concept (semantically equivalent) if one *object* is an abbreviation modifier of the other *object*.
5. If an *object* is a numeric modifier of another *object*, then it specifies a *property* of the *object*.

3.2. Automatic Construction of Requirements Ontology

The requirements elements are attached into the base ontology in order to automatically construct requirements ontology of each stakeholder as a common representation for knowledge interchange purposes. In our work, we represent each construct in the ontologies by using the first-order logic (FOL) representation for machine-readable and the visualization view for the users. The graphical notations of the visualization view are defined in Figure 2. Note that some notations are enhanced from unified modeling language (UML)-like but they have more specific meaning used in this work.

A base ontology, constructed by ontology engineer using *IEEE Std 830-1998* [23] and *ESA PSS-05-03* [24], can be represented in FOL and graphical representation as illustrated in Figure 3. It contains requirements types of requirements artifacts in the domain of software requirements. We use a top-down approach [28] in order to construct a base ontology starting from creating the most general concept of *requirement artifact* which is classified into two

concepts: *functional requirement* and *non-functional requirement*. The *functional requirement* concept can be further classified as *data specification*, *process specification* and *control specification*. We apply the same principle to categorize other concepts in the base ontology.

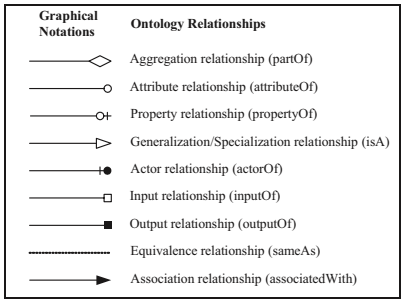


Figure 2. Graphical Notations of Ontology Relationships

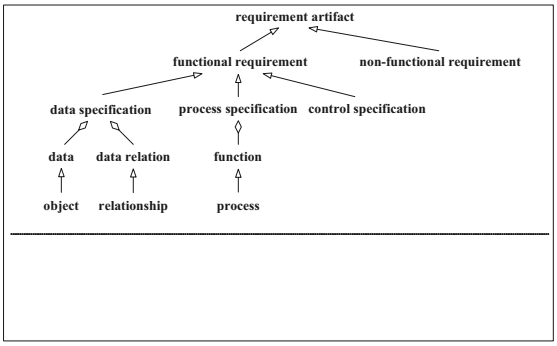


Figure 3. A Graphical Representation of a Base Ontology

3.3. Automatic Generation of Traceability Relationships

We match the concepts between two requirements ontologies by applying S-Match [18] approach with the reason as mention earlier in Section 2. However, we extend it to cover nine kinds of ontology relationships and to produce *overlapping* relation. We also allow the users to incorporate domain knowledge for resolving lack of background knowledge found in S-Match. The proposed base ontology is used to reduce the number of all possible matching concepts for increasing the matching performance.

The S-Match algorithm is organized in four macro steps described in [18]. For element matching (step 3), we use external resources (i.e., domain knowledge, WordNet [29]) and string matching techniques (i.e., prefix, suffix, edit distance, n-gram) with threshold 0.85. Lexical relations provided by WordNet are converted to semantic relations according to the rules as shown in Table 1.

Table 1. Conversion of lexical relations to semantic relations

Lexical Relations	Semantic Relations
Synonym	$a = b$
Hypernym or holonym	$a \supseteq b$
Hyponym or meronym	$a \subseteq b$
Antonym	$a \perp b$

Each concept is converted into a propositional validity problem. Semantic relations are translated into propositional connectives using the rules described in Table 2. We extend the *overlapping relation* in the last row of the table.

Table 2. The relationships between semantic relations and propositional formula

Semantic Relations	Propositional Logic	Translation of Formula into Conjunctive Normal Form
$a = b$	$a \leftrightarrow b$	axioms $\wedge \forall(context_1 \wedge \neg context_2)$ axioms $\wedge \forall(\neg context_1 \wedge context_2)$
$a \subseteq b$	$a \rightarrow b$	axioms $\wedge \forall(context_1 \wedge \neg context_2)$
$a \supseteq b$	$b \rightarrow a$	axioms $\wedge \forall(\neg context_1 \wedge context_2)$
$a \perp b$	$\neg(a \wedge b)$	axioms $\wedge \forall(context_1 \wedge context_2)$
$a \cap b$	$(a \wedge b) \vee$ $(a \wedge \neg b) \vee$ $(\neg a \wedge b)$	axioms $\wedge \exists(\neg context_1 \vee \neg context_2) \wedge$ $\exists(\neg context_1 \vee context_2) \wedge$ $\exists(context_1 \vee \neg context_2)$

The criterion for determining whether a relation holds between the concepts is the fact that it is entailed by the premises. Thus, we have to prove that this formula *(axioms)* $\rightarrow$ *rel(context$_1$, context$_2$)* is valid. A propositional formula is valid iff its negation is unsatisfiable. A SAT solver [30] run on the formula fails.

We use types of overlap relations defined in [31] to generate traceability relationships in our work. The traceability relationships can be generated when a match is found in the ontologies. Thus, the semantic relations will be mapped to traceability relationships as shown in Table 3.

Table 3. Conversion of semantic relations into traceability relationships

Semantic Relations	Traceability Relationships
Equivalence (=)	overlapTotally (=)
More or less general ($\supseteq$,$\subseteq$)	overlapInclusively ($\supseteq$,$\subseteq$)
Mismatch ($\perp$)	noOverlap ($\perp$)
Overlapping ($\cap$)	overlapPartially ($\cap$)

The distinction among different types of traceability relationships is important because these have a different impact on the requirements traceability status of two requirements artifacts. More specifically, the ontology matcher module discards *noOverlap* relationship in this work because there is no effect on multiperspective requirements artifacts changes.

4. An Illustrative Example: Multiperspective Requirements Traceability

This section provides an example to illustrate how the proposed MPRT framework can solve the heterogeneity problems found in multiperspective requirements artifacts efficiently. We demonstrate that two different stakeholders (*System User 1* and *2*) produce *Doctor Investigation System (DIS)* and *In-Patient Registration System (IPRS) Requirements* respectively as depicted in Figure 4. They want to collaborate with each other by sharing the requirements of two overlapping systems which are parts of a hospital information system. These requirements are overlapping and generated with respect to different perspectives. Multiple sets of vocabularies may be used in the requirements descriptions.

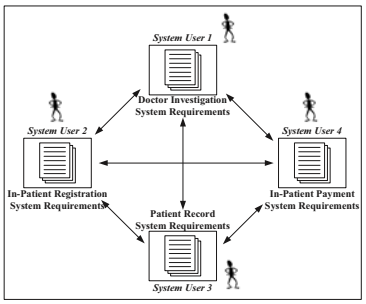


Figure 4. Distributed Collaboration of Multiperspective Requirements Artifacts in a Hospital Information System

Assume that a system user 1 of DIS expresses his/her requirements as follows:

> *Each patient has a unique hospital number (HN) and a name. A patient is admitted by a doctor. Nurses and doctors are considered as staffs. A nurse has a name. The nurse's name consists of a first name, an initial and a last name. A doctor is identified by an identification number and a name.*

A system user 2 of IPRS defines his/her requirements in the following.

> *Physicians and nurses are staffs. Staffs have an ID, a name and an address. A surgeon is a physician.*

Both requirements are presented as a source (DIS) and a target (IPRS) in our *MPRT browser*. After both requirements are passed to requirements analyzer and REG modules, the ROC module will attach requirements elements into the base ontology. Accordingly, the DIS and IPRS requirements ontology are automatically constructed as depicted in Figure 5 (exclude a base ontology).

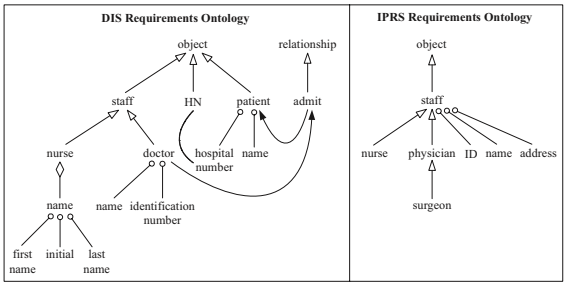


Figure 5. DIS and IPRS Requirements Ontology

To check the unsatisfiability of a propositional formula done by ontology matcher module we use the standard DPLL-based SAT solver [30]. From the example in Figure 5, trying to prove that $doctor_1$ in DIS requirements ontology is less general than $physician_2$ in IPRS requirements ontology, requires constructing the following formula.

$$((staff_1 \leftrightarrow staff_2) \wedge (doctor_1 \leftrightarrow physician_2)) \wedge (staff_1 \wedge doctor_1) \wedge \neg(staff_2 \wedge physician_2)$$

The above formula turns out to be unsatisfiable, and therefore, the less general relation holds. It is noticeable that if we test for the more general relation between the same pair of concepts, the corresponding formula would be also unsatisfiable. As a result, the final relation for the given pair of concepts is the equivalence.

Some parts of traceability relationships between DIS and IPRS requirements ontology can be automatically generated by ontology matcher module as shown in the predicate terms below.

overlapTotally($staff_1$, $staff_2$)
overlapTotally($staff_1$/$nurse_1$, $staff_2$/$nurse_2$)
overlapTotally($staff_1$/$doctor_1$, $staff_2$/$physician_2$)
overlapTotally($staff_1$/$nurse_1$/$name_1$, $staff_2$/$nurse_2$/$name_2$)
overlapTotally($staff_1$/$doctor_1$/$name_1$, $staff_2$/$physician_2$/$name_2$)
overlapTotally($staff_1$/$doctor_1$/identification_$number_1$, $staff_2$/$physician_2$/ID_2)
overlapPartially($staff_1$/$nurse_1$, $staff_2$/$physician_2$)
overlapPartially($staff_1$/$doctor_1$, $staff_2$/$nurse_2$)
overlapInclusively($staff_1$/$nurse_1$/$name_1$/first_$name_1$, $staff_2$/$nurse_2$/$name_2$)
overlapInclusively($staff_1$/$nurse_1$/$name_1$/last_$name_1$, $staff_2$/$nurse_2$/$name_2$)

5. Conclusions and Ongoing Work

In this paper, we point out the heterogeneity problems found in multiperspective requirements artifacts. *Requirements ontology* can be used as a knowledge management mechanism to describe these artifacts in a common way to share understanding among various stakeholders for interoperability and traceability purposes. The traceability relationships can be automatically generated by using ontology matching

technique. Hence, the combination of NLP techniques, a rule-based approach, an automated reasoning process and ontology concepts provides a practical and powerful multiperspective requirements traceability mechanism.

Currently, our MPRT framework emphasizes on requirements artifacts represented in terms of natural language or text but we can extend a base ontology to cover software artifacts in other phases of software development process.

Acknowledgements

The first author is supported by the grant from the Department of Computer Science at Mahidol University. Also I would like to acknowledge University of the Thai Chamber of Commerce for their financial support for this presentation.

6. References

[1] "SE Tools Taxonomy - Requirements Traceability Tools", International Council on Systems Engineering (INCOSE), Available at http://www.incose.org/productspubs/products/setools/tooltax/reqtrace_tools.html, September 22, 2004.

[2] K. Pohl, "PRO-ART: Enabling Requirements Pre-Traceability", *Proceedings of the 2nd International Conference on Requirements Engineering (ICRE'96)*, Colorado Springs, Colorado, U.S.A., April 15-18, 1996, pp. 76-84.

[3] J.C. Huang, C.K. Chang, and M. Christensen, "Event-Based Traceability for Managing Evolutionary Change", *IEEE Transactions on Software Engineering*, Vol.29, No.9, September, 2003, pp. 796-810.

[4] S.A. Sherba, M.A. Kenneth, and M. Faisal, "A Framework for Mapping Traceability Relationships", *Proceedings of the Second International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'03), In Conjunction with the 18th IEEE International Conference on Automated Software Engineering*, Montreal, Quebec, Canada, October 7, 2003.

[5] G. Gudgeirsson, "Requirements Engineering and XML", September 22, 2000.

[6] J. Wu and J. Han, "XmlTRAM+: Using XML Technology to Manage Software Requirements and Architectures", *Proceedings of the 8th Australian World Wide Web Conference*, Twin Waters Resort, Sunshine Coast, Australia, July, 2002, pp. 237-245.

[7] G. Spanoudakis et al., "Rule-Based Generation of Requirements Traceability Relations", *Journal of Systems and Software*, Vol.72, No.2, 2004, pp. 105-127.

[8] E. Rahm and P.A. Bernstein, "A Survey of Approaches to Automatic Schema Matching", *The International Journal on Very Large Data Bases (VLDB)*, Vol.10, No.4, 2001, pp. 334-350.

[9] P. Shvaiko and J. Euzenat, "A Survey of Schema-Based Matching Approaches", *Journal on Data Semantics IV*, 2005, pp. 146-171.

[10] L. Palopoli, D. Sacca, and D. Ursino, "Semi-Automatic, Semantic Discovery of Properties from Database Schemes", *Proceedings of International Database Engineering and Applications Symposium (IDEAS'98), IEEE Computing*, July 8-10, 1998, pp. 244-253.

[11] T. Milo and S. Zohar, "Using Schema Matching to Simplify Heterogeneous Data Translation", *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB)*, 1998, pp. 122-133.

[12] P. Mitra, G. Wiederhold, and J. Jannink, "Semi-Automatic Integration of Knowledge Sources", *Proceedings of the 2nd International Conference on Information Fusion'99*, Sunnyvale, U.S.A., July, 1999.

[13] S. Bergamaschi, S. Castano, and M. Vincini, "Semantic Integration of Semistructured and Structured Data Sources", *ACM SIGMOD Record*, Vol.28, No.1, 1999, pp. 54-59.

[14] J. Madhavan, P.A. Bernstein, and E. Rahm, "Generic Schema Matching with Cupid", *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, Roma, Italy, 2001, pp. 49-58.

[15] H.H. Do and E. Rahm, "COMA - A System for Flexible Combination of Schema Matching Approaches", *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, Hong Kong, China, 2002, pp. 610-621.

[16] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching", *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, San Jose, C.A., 2002, pp. 117-128.

[17] P. Bouquet, L. Serafini, and S. Zanobini, "Semantic Coordination: A New Approach and An Application", *Proceedings of International Semantic Web Conference (ISWC)*, 2003, pp. 130-145.

[18] F. Giunchiglia, M. Yatskevich, and P. Shvaiko, "Semantic Matching: Algorithms and Implementation", *Journal on Data Semantics IX*, 2007, pp. 1-38.

[19] D. Aumuller et al., "Schema and Ontology matching with COMA++", *Proceedings of International Conference on Management of Data (SIGMOD), Software Demonstration*, Baltimore, Maryland, U.S.A., June 14-16, 2005.

[20] K.C. Chang, B. He, and Z. Zhang, "Toward Large Scale Integration: Building a Metaquerier Over Databases on the Web", *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2005, pp. 44-55.

[21] S. Castano, A. Ferrara, and S. Montanelli, "Matching Ontologies in Open Networked Systems: Techniques and Applications", *Journal on Data Semantics V*, 2006, pp. 25-63.

[22] "The Stanford Parser: A Statistical Parser (version 1.6)", Stanford University, Available at http://nlp.stanford.edu/software/lex-parser.shtml, August 18, 2007.

[23] "IEEE Recommended Practice for Software Requirements Specifications", *IEEE Std 830-1998*, The Institute of Electrical and Electronics Engineers (IEEE), June 25, 1998.

[24] "Guide to the Software Requirements Definition Phase", *ESA PSS-05-03*, European Space Agency (ESA), Issue 1, Revision 1, March, 1995.

[25] H.M. Harmain and R. Gaizauskas, "CM-Builder: A Natural Language-Based CASE Tool for Object-Oriented Analysis", *Automated Software Engineering (ASE)*, Vol.10, No.2, April, 2003, pp. 157-181.

[26] J. Wielemaker, "SWI-Prolog version 5.6.30", University of Amsterdam, Available at http://www.swi-prolog.org/, 1990-2007.

[27] M.-C. de Marneffe, B. MacCartney, and C.D. Manning, "Generating Typed Dependency Parses from Phrase Structure Parses", *5th International Conference on Language Resources and Evaluation (LREC 2006)*, 2006.

[28] N.F. Noy and D.L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology", *Technical Report*, Stanford Knowledge Systems Laboratory, March, 2001.

[29] G.A. Miller, "WordNet: A Lexical Database for English", *Communications of the ACM*, Vol.38, No.11, November, 1995, pp. 39-41.

[30] D.L. Berre, "A Satisfiability Library for Java", Available at http://www.sat4j.org.

[31] G. Spanoudakis, A. Finkelstein, and D. Till, "Overlaps in Requirements Engineering", *Automated Software Engineering*, Vol.6, No.2, April, 1999, pp. 171-198.

Eliciting Scenarios from Scenarios

Abdolmajid Mousavi and Behrouz H. Far
Department of Electrical and Computer Engineering
University of Calgary
2500 University Drive N.W., T2N1N4
Calgary, AB, Canada
amousavi@ucalgary.ca, far@ucalgary.ca

ABSTRACT
Scenario-based software development has been widely accepted for the description of concurrent systems. However, scenario-based software development is inherently partial as scenarios can only show instances of the system behaviour and because the behaviour of each component is only meaningful in collaboration with other components in scenarios. Therefore, sound, formal, and structured approaches which help in eliciting scenarios that cover more aspects of the system requirements are welcome in software engineering.

This paper presents an approach for incorporating the domain knowledge and the system architecture depicted in scenarios for elaborating scenario specifications. Our approach can be repeatedly applied in a cycle that consists of eliciting the domain knowledge from scenarios and producing new scenarios until a satisfactory specification is reached.

KEY WORDS
Scenario-based specifications, domain knowledge, emergent scenarios

1. Introduction

Scenario-based languages such as Message Sequence Charts (MSCs) [1], are one of the popular ways for the description of concurrent systems because of the desire of stakeholders to describe system's functionality by small and partial stories. However, scenario are only instances of the system behaviour and might not cover all the system requirements [7], [13]. Furthermore, the behaviour of each system component is only meaningful in conjunction with other components. This means that a mechanism is needed for composing component's behaviours in each scenario in order to build behaviour models for components [5], [14], [15].

In this paper, we present an approach for incorporating the domain knowledge and the system architecture depicted in scenarios for elaborating scenario specifications. The required domain knowledge will be obtained by referring to an available (and possibly incomplete) set of scenarios. The domain knowledge and the available set of scenarios are used for the synthesis of behaviour models for system's components (also called processes). Then, emergent behaviours obtained from this synthesis process are converted into scenarios that can be added to the current set of scenarios and enrich the system specification.

The advantage of our approach can be better envisaged in the general software development practice of Figure 1. In this figure, first behaviour models are constructed from a scenario specification as an initial approximation of system. Then, an analysis phase begins in which any mismatch between the scenario specification and behaviour models are found in terms of emergent behaviours. After being detected, emergent behaviours will be validated against system goals and properties in order to provide the required feedback for the system analyst to correct the specification. The process of correction and analysis continues until a satisfiable specification is obtained.

2. Related Work

Recently, there has been a growing body of research around scenario-based software development with different motives, assumptions and methodologies. Nevertheless, we can sketch some boundaries for different groups of work -though with overlap - in this research area.

The first group assumes that while a scenario specification is complete with respect to the behaviour of individual processes, it might not be complete with respect to the behaviour of system [2], [11], [13]. Some common problems addressed by this research area are deadlocks, implied scenarios, and safe or weak realizability. Our work in [10], which introduces the notion of strong safe realizability belongs to this category.

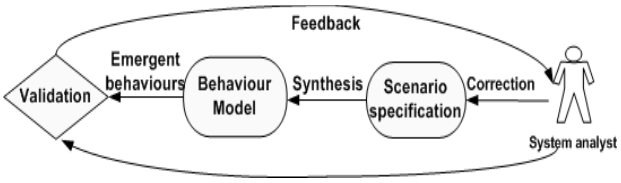


Figure 1. A framework for elaborating scenarios.

The second group assumes that a scenario description might not be complete with respect to the behaviour of individual processes, and therefore in the first place, they are interested in building the right behaviour models for processes using scenarios and the domain knowledge [7], [9], [14], [15]. The assumption here is that the scenario specification is describing a state machine that models the behaviour of system components. Thus, components in scenarios are considered to model both a set of states in the state machine (which are called component states) and the events that fire state change (called labeled transitions). Therefore, the same component states in different scenarios provide information of how the state machines from different scenarios should be combined.

In the current practice of synthesis of behaviour models, similar states in different state machines will be merged in order to obtain a single state machine for the behaviour of each component [3], [5], [6], [8], [9], [12], [14], [15]. While many of these approaches use the domain knowledge in addition to scenarios, they do not give a formal and clear picture of the domain knowledge needed for their purpose. Furthermore, because they do not consider the system architecture defined by scenarios, they usually result in spurious emergent behaviours for the components (and system) (a phenomenon that is also called overgeneralization). In contrast, in our approach the required domain knowledge is clearly defined and emergent behaviours occur only if the system architecture allows for them.

3. Basic Definitions

We assume that scenarios are represented by Message Sequence Charts defined as follows. Let P be a finite set of processes (system's components) with the total number of processes $|P| \geq 2$, and C be a finite set of message contents (or message labels). Denote $\Sigma_i = \{i!l(c), i?l(c) | l \in P \setminus \{i\}, c \in C\}$ to be the alphabet of process $i \in P$, where $i!l(c)$ denotes an event that sends a message from process i with content c to process l, whereas $i?l(c)$ denotes an event that receives on process i a message with content c from process l. Also, the alphabet (of all processes $i \in P$) will be $\Sigma = \bigcup_{i \in P} \Sigma_i$.

Definition 1 *(Message Sequence Chart): A Message Sequence Chart (MSC) over P and C is defined to be a tuple $m = (E, \alpha, \beta, \prec)$ where:*
- E is a finite set of events.
- $\alpha : E \rightarrow \Sigma$ maps each event to its label. The set of events located on process i is $E_i = \alpha^{-1}(\Sigma_i)$. The set of all send events in the event set E is denoted by $E! = \{e \in E | \exists i, l \in P, c \in C : \alpha(e) = i!l(c)\}$ and the set of receive events as $E? = E \setminus E!$.
- $\beta : E! \rightarrow E?$, is a bijection mapping between send and receive events such that whenever $\beta(e_1) = e_2$ and $\alpha(e_1) = i!l(c)$, then $\alpha(e_2) = l?i(c)$.
- $\prec$ is a partial order on E such that for every process $i \in P$, the restriction of $\prec$ to E_i is a total order, and $\prec$ is equal to the transitive closure of $\{(e_1, e_2) | e_1 \prec e_2, \exists i \in P : e_1, e_2 \in E_i\} \cup \{(e, \beta(e)) | e \in E!\}$.

Fig. 2 shows a set of scenarios for an ATM machine in MSC notation.

We also define the projection $m|_i$ for process i in MSC m to be the ordered sequence of messages that corresponds to the events occurring on process i in the MSC m. For $m|_i$, $\|m|_i\|$ indicates its length, which is equal to the total number of events of m on process i, and $m|_i[j]$ refers to j^{th} element of $m|_i$, so that if e_j is the jth event on process i according to the total order of the events of i in m, then $\alpha_m(e_j) = m|_i[j-1], 0 < j < \|m|_i\|$.

Definition 2 *(equivalent Finite State Machine of a projection): For the projection $m|_i$, we define an equivalent FSM (eFSM) $A_i^m = (S^m, \Sigma_i, \delta^m, q_0^m, q_f^m)$ such that:*
- $S^m = \left\{q_0^m, \cdots, q_f^m\right\}$ is a finite set of states
- Σ_i is the alphabet
- q_0^m is the initial state
- $q_f^m = q_{\|m|_i\|}^m$ is the final state (accepting state)
- δ^m is the transition relation such that $\delta(q_j^m, m|_i[j]) = q_{j+1}^m$, $0 \leq j < f$, and the only word accepted by A_i^m is $m|_i$.

For instance, the eFSM of the projection of scenario m_2 in Fig. 2 on ATM process is shown in Fig. 3 in which $q_0^{m_2}$ is its initial state and $q_{13}^{m_2}$ is its final state.

4. From Scenarios to Domain Knowledge

4.1. Semantical Causality

Assuming that scenarios do not provide all the necessary behaviours for processes, the domain knowledge will be needed for building the right behavioral models for processes [7], [9], [15]. In this regard, a distinct feature of our approach is that it defines the domain knowledge based

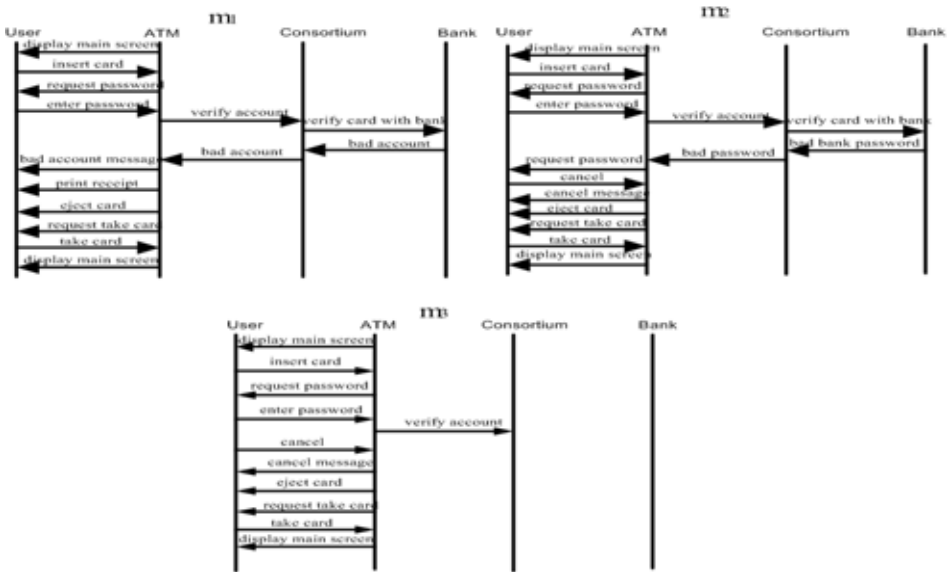


Figure 2. A preliminary set of scenarios for an ATM system.

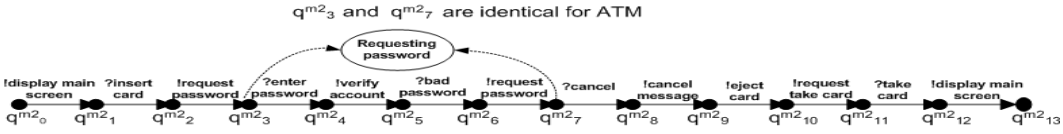


Figure 3. A state machine (eFSM) for ATM extracted from m2.

on a fixed relation between messages in scenarios called *semantical causality*. Semantical causality captures performance dependability between messages as a part of the domain knowledge that is not explicitly defined in scenarios.

Definition 3 *(Semantical causality): We say message* $m|_i[j]$ *is a semantical cause for message* $m|_i[k]$ *and denote it by* $m|_i[j] \overset{se}{\rightarrow} m|_i[k]$ *if process* i *has to keep the result of the operation of* $m|_i[j]$ *in order to perform* $m|_i[k]$.

Similar to [7], by the operation of a message we mean the ultimate purpose of the message. For example, the corresponding operation for the *insert card* message for the ATM is: *card is inserted*. Note that, semantical causality comes from the domain knowledge and can be found without referring to ordering of messages in scenarios. Also, note that semantical causality is an invariant property for a system because it is the system's architecture and the domain knowledge that dictate whether or not one message is needed by a process in order to perform another message.

For example, in m_2, *insert card* is a semantical cause for *eject card* because ATM has to keep the card inserted before it can eject the card. As another example, in a lift system, the message *close door* is a semantical cause for the message *open door* or the message *lift moving* because the lift has to keep the door closed before it can open the door or before it can move.

4.2. State Values and Identical States

The main problem in behaviour modeling is how to detect identical states for a process in different eFSMs extracted from scenarios. The work of [15] uses global system variables to mark states that a process goes through as it is communicating with other processes in a scenario. However, different domain experts can choose different system variables. Consequently, different behavioural models for a process might be obtained for a given application and at the end it is not clear which model is the right one. Note that, this problem is the result of **choosing** different variables not updating the values of variables (in fact, an implicit assumption in [15] is that the domain expert makes no mistakes in updating variables). To overcome this drawback, we let the current state of the process in any eFSM to be defined by the messages that the process needs them in order to perform the messages that come after its current state. Considering Definition 3, these are the messages that are semantical causes for the messages in the transitions after the current state of the process in the eFSM.

More specifically, we associate a *state value* $v_i(q^m_k)$ to

every state q_k^m in the eFSM A_i^m, $i \in P$, $m \in M$ as follows.

Definition 4 *(State value): The state value $\upsilon_i(q_k^m)$ for the state q_k^m in eFSM $A_i^m = (S^m, \Sigma_i, \delta^m, q_0^m, q_f^m)$ is a word over the alphabet $\Sigma_i \cup \{1\}$ such that $\upsilon_i(q_0^m) = 1$, $\upsilon_i(q_f^m) = m|_i[f-1]$, and for $0 < k < f$ is defined as follows:*
i) $\upsilon_i(q_k^m) = m|_i[k-1]\upsilon_i(q_j^m)$, if there exist some j and l such that j is the maximum index that $m|_i[j-1] \stackrel{se}{\rightarrow} m|_i[l]$, $0 < j < k$, $k \leq l < f$
ii) $\upsilon_i(q_k^m) = m|_i[k-1]$, if Case i) does not hold but $m|_i[k-1] \stackrel{se}{\rightarrow} m|_i[l]$, for some $k \leq l < f$
iii) $\upsilon_i(q_k^m) = 1$, if none of the above cases hold

In Definition 4, first state values of the initial and the final states of an eFSM are defined. Then, the states value of the state q_k^m is defined depending on whether for the transitions that come after this state: there exists a message $m|_i[j-1]$ as a semantical cause, $0 < j < k$ (Case i)), or $m|_i[k-1]$ is the only semantical cause (Case ii)), or neither $m|_i[k-1]$ nor any other message is a semantical cause (Case iii)). In particular, the last case marks all the states q_k^m that from the processes perspective are like its initial state with the state value of 1.

Since for a given application and process, semantical causality between messages is an invariant property defined by the domain knowledge, state values of the states of a process are independent of the choice of the domain expert.

Let's calculate state values of states q_3^{m2} and q_7^{m2} in Fig. 3. From the domain knowledge pertinent to ATM system, the maximum index j for which $m2|_{ATM}[j-1]$ is a semantical cause for a message in the transitions after q_3^{m2} is $j = 2$ for which $m2|_{ATM}[2-1] =$*insert card* (for instance *insert card* $\stackrel{se}{\rightarrow}$ *eject card*). Thus, based on Case i) of Definition 4 we have: $\upsilon_{ATM}(q_3^{m2}) = m2|_{ATM}[3-1]\upsilon_{ATM}(q_2^{m2})$. To calculate $\upsilon_{ATM}(q_2^{m2})$, observe that *insert card* is the only semantical cause for messages after q_2^{m2}, and therefore, based on Case ii) of Definition 4, we have $\upsilon_{ATM}(q_2^{m2}) =$*insert card*. Thus, $\upsilon_{ATM}(q_3^{m2}) =$ *(request password) (insert card)*. For q_7^{m2} because none of the messages *bad password*, *verify account*, and *enter password* is a semantical cause for the messages in the transitions after q_7^{m2}, still the maximum index j would be $j = 2$ for which $m2|_{ATM}[2-1] =$*insert card*. Thus, with the same reasoning as for q_3^{m2}, we have: $\upsilon_{ATM}(q_7^{m2}) = m2|_{ATM}[7-1]\ \upsilon_{ATM}(q_2^{m2}) =$ *(request password) (insert card)*. This means that from ATM's perspective, two states $q_3^{m_2}$ and $q_7^{m_2}$ are identical in Figure 3.

An interesting case is the state values of the user. Since the user does not need any message in order to perform other messages, there would be no semantical causality between messages for the user. Thus, according to Case iii) of Definition 4, all the states of the user except its final state are identical with the state value of 1. This is the result of the fact that no protocol is imposed on the user to communicate with ATM (see [4]).

5. From Domain Knowledge to Scenarios

5.1. Criteria for Merging Identical States

The common practice in the synthesis of behaviour models from scenarios is to merge identical states of processes [5], [6], [9], [14], [15]. However, this is a blind process since it can create many spurious emergent behaviours in the resulting behaviour models [5], [9]. One way to reduce this effect is to first check whether the emergent behaviour that could be generated as the result of merging states is allowed by the system architecture defined in scenarios. By system architecture, we mean the processes, the partial order between events in scenarios, and the available information regarding process's identical states (see Figure 3).

Figure 4 shows a general case where two identical states q_i and q_j of two state machines (eFSMs $A_k^m = A$ and $A_k^n = B$ obtained from scenarios m and n) for the process k are merged into a single state q. a_i, a_{i+1},..., are the send or receive messages for the process from scenario m, whereas b_j, b_{j+1}, ..., are are the send or receive messages for the process from scenario n. Thus, $...a_i$ shows a sequence of send and receive messages that ends in a_i and $b_{i+1}...$ shows a sequence that starts with b_{i+1}.

A possible emergent behaviour in Figure 4 is the sequence $...a_ib_{j+1}...$ (or $...b_ja_{i+1}...$) where $...a_i$ shows a behaviour from state machine A whereas $b_{j+1}...$ is a behaviour from state machine B. Thus, the possible emergent behaviour $...a_ib_{j+1}...$ is obtained from a combination of a behaviour from A with another one from B. Now, we shall look for a set of criteria under which $...a_ib_{j+1}...$ is possible. The intention behind these criteria is to avoid generalization (merging identical states) unless we have enough evidence in scenarios.

Having this said and depending on whether b_{j+1} is a send or receive message for k, to have $...a_ib_{j+1}...$ as the result of merging q_i and q_j, one of the followings cases must hold:

i) b_{j+1} is a send message for the process k. Therefore, nothing can prevent k to initiate sending b_{j+1} when it is in state q and generate the emergent behaviour $...a_ib_{j+1}...$

ii) k stops after b_j. In other words, q_j is a final state for B. In this case if a_{i+1} is a send message for k, then k has initiative to send (because of the state machine A) or stop to send (because of the state machine B) a_{i+1} when it is in state q resulting in (when it stops sending a_{i+1}) the emergent behaviour $...a_i$ i.e. it stops after sending $...a_i$ while according to the scenario m and the state machine A, it must continue with message a_{i+1}

iii) b_{j+1} is a receive message for k and in scenario

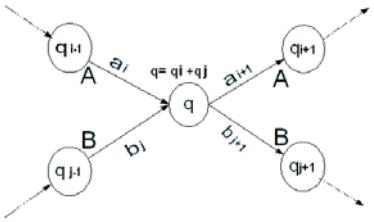


Figure 4. Two states q_i and q_j of state machines A (eFSM A_k^m) and B (eFSM A_k^n) are merged.

m, another process say k', can send b_{j+1} to k even when a_{i+1} does not happen for k, and in m, k receives b_{j+1} after a_{i+1}. In this case, again the emergent behaviour $...a_ib_{j+1}...$ can happen

Criteria i) and ii) represent conditions where the process has initiative either to send or stop sending a message, whereas criterion iii) represents conditions over two processes involved in sending and receiving a message. This latter case can be justified using our basic rule outlined before, that is: we look for any evidence in scenarios that allows for emergent behaviours. Because b_{j+1} is a receive message for the process, we should look for an evidence that shows the process is able to receive b_{j+1} after a_i happens for it. Therefore, first we must make sure that in m there is a process (k') other than k that sends b_{j+1} to k. Second, we must make sure that k' can send b_{j+1} to k even when a_{i+1} does not happen for k because is a_{i+1} is a necessary condition to have b_{j+1} sent by k', then by removing a_{i+1}, k' will not be able to send b_{j+1}. As a result, k will not be able to receive b_{j+1}, and so, it would be impossible for the emergent behaviour $...a_ib_{j+1}...$ to happen for k. Also, the requirement for receiving of b_{j+1} after a_{i+1} is to ensure that in m, b_{j+1} is not consumed by k before a_{i+1} otherwise there would be impossible to have the emergent behaviour $...a_ib_{j+1}...$ and b_{j+1} would be simply one of the messages in the sequence $...a_i$.

Note that since the aforementioned criteria are defined over scenarios, they can be automatically checked using syntactic constructs employed in Definition 1 (processes, events and messages, and partial order between events), and identical states. More specifically, assuming that identical states of processes are available, for criteria i) or ii) we need respectively to check whether or not a message is a send message or a state is a final state for a given process. For Case iii), we need to check whether a process is receiving a message in a sequence diagram and does the partial order between events of the sequence diagram or the state information of another process that sends the message allows for receiving the message.

5.2. Emergent Scenarios

In this section we show how the formalized domain knowledge along with the criteria for merging identical states discussed in the previous section result in emergent scenarios. Consider Figures 3 and its two identical states ($q_3^{m_2}$ and $q_7^{m_2}$ with the same state of requesting password). Since both messages after these states are receive messages for ATM, Case iii) of our criteria needs to be checked. In other words, it should be checked whether or not the process (user) that sends these messages to ATM is able to send them.

The outgoing transitions from states $q_3^{m_2}$ and $q_7^{m_2}$ are respectively the *enter password* and the *cancel* messages, which are two send events for the user that occur after a pair of its identical states (remember from Section 4.2 that all the states of the user have the state value of 1 except its final state). Therefore, Case i) applies to the user and the states after two request password will be merged for the user. This means that the user can either send enter password or cancel after receiving request password. Therefore, ATM also can either receive enter password or cancel after sending request password. Thus, Case iii) applies for ATM for states $q_3^{m_2}$ and $q_7^{m_2}$ and these states can be merged. As the result of this merge, the state machine of Figure 5a) will be obtained from Figure 3 as a part of behaviour model for ATM.

Figure 5a) shows an emergent behaviour for ATM (indicated as the darker path in the figure), which is shown in Figure 5b) in terms of a scenario. This is a valid scenario for ATM that is ignored in the original specification given by m_1, m_2, and m_3. The system analyst (see Figure 1) can enrich the scenario specification for the ATM system by adding the scenario of Figure 5b) as the fourth scenario to Figure 2 and starts a new cycle of synthesis-emergent behaviours-correction.

6. Conclusions and Future Work

Our approach for eliciting scenarios is a two steps cycling process. In the first step, scenarios are used for eliciting the domain knowledge. In the second step, the acquired domain knowledge is used along with the system architecture depicted in scenarios to infer emergent scenarios that cover overlooked system's aspects. As it is shown in Figure 1, by cycling through these steps we hope to end up in a more complete specification.

For future work, scalability of our approach is an issue that needs to be verified on systems in different domains and with various complexities, particularly because finding semantical causality between messages might be difficult when the application domain or size is changed.

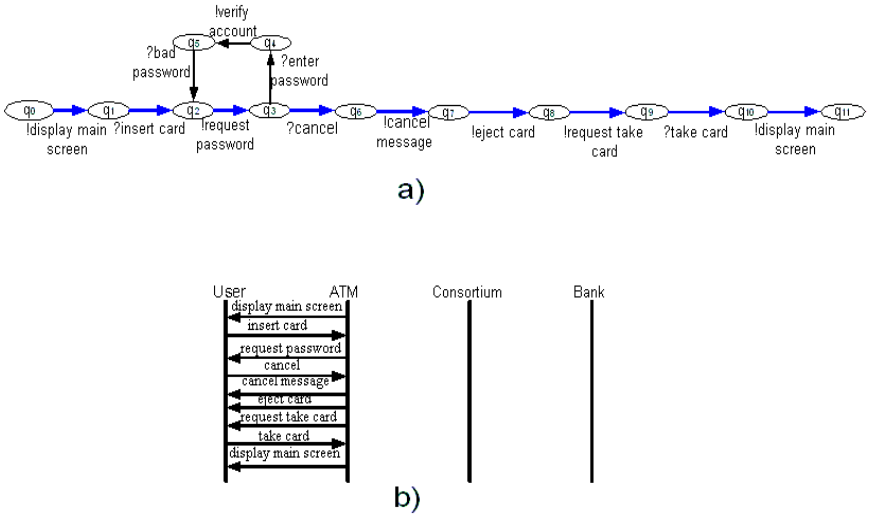


Figure 5. a) A state machine obtained from Figure 3 when merging states $q_3^{m_2}$ and $q_7^{m_2}$; b) An emergent behaviour for ATM that can be added to the set of scenarios of Figure 2.

References

[1] *Recommendation Z.120: Message Sequence Charts (MSCs)*. Geneva., 1996.

[2] R. Alur, K. Etessami, and M. Yannakakis. Inference of Message Sequence Charts. *IEEE Trans. on Software Eng.*, 29(7):623–633, July 2003.

[3] D. Harel and H. Kugler. Synthesizing state-based object systems from lsc specifications. *International Journal of Foundations of Computer Science*, 13(1):5–51, 2002.

[4] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001.

[5] K. Koskimies and E. Mäkinen. Automatic synthesis of state machines from trace diagrams. *Software Practice and Experience*, 24(7):663–658, 1994.

[6] I. Krüger, R. Grosu, P. Scholz, and M. Broy. From MSCs to statecharts. In: Franz J. Rammig (ed.): Distributed and Parallel Embedded Systems, Kluwer Academic Publishers, 1999.

[7] A. v. Lamsweerde and W. L. Inferring declarative requirements specifications from operational scenarios. *IEEE Trans. on Software Eng.*, 24(12):1089–1114, December 1998.

[8] S. Leue, L. Mehrmann, and M. Rezai. Synthesizing ROOM models from Message Sequence Charts specifications. In *Proc. of 13th IEEE Conf. on Automated Sofware Eng. (ASE '98)*, pages 192–195, 1998.

[9] E. Mäkinen and T. Systä. MAS - an interactive synthesizer to support behavioral modeling in uml. In *ICSE 2001*, pages 15–24, Toronto, May 2001.

[10] A. Mousavi, B. Far, A. Eberlein, and B. Heidari. Strong safe realizability of Message Sequence Chart specifications. In *International Symposium on Fundamentals of Software Engineering (FSEN)*, pages 334–349. LNCS 4767, April 2007.

[11] H. Muccini. Detecting implied scenarios analyzing non-local branching choices. In *FASE 2003*, Warsaw, Poland, april 2003.

[12] S. Somé, R. Dssouli, and J. Vaucher. From scenarios to timed automata: Building specifications from user requirements. In *Proc. Asia Pacific Software Eng. Conf. (APSEC '95)*, pages 48–57, 1995.

[13] S. Uchitel. *Incremental Elaboration of Scenario-Based Specifications and Behaviour Models Using Implied Scenarios*. PhD thesis, Imperial College, Londonn, 2003.

[14] S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavioral models from scenarios. *IEEE Trans. on Software Eng.*, 29(2):99–115, February 2003.

[15] J. Whittle and J. Schumann. Generating statecharts designs from scenarios. In *ICSE 2000*, Limerick, Irland, 2000.

Tailoring an Aspectual Goal-Oriented Approach to Model Features[ζ]

Carla Silva[*], Fernanda Alencar
Univ. Federal de Pernambuco
Recife-PE, Brasil, 50732-9
ctlls@cin.ufpe.br, fmra@ufpe.br

João Araújo, Ana Moreira
CITI/FCT, Univ. Nova de Lisboa
Caparica, Portugal, 2829-516
{ja, amm}@di.fct.unl.pt

Jaelson Castro
Centro de Informática
Univ. Federal de Pernambuco
jbc@cin.ufpe.br

Abstract

A feature model can represent commonalities and variabilities in software product lines, but it does not describe how these features are achieved through system functionality and how these functionalities work together to achieve the expected system behavior. Moreover, feature models lack foundations to reason about the relationships among different requirements. Thus, it makes it difficult to justify why a specific feature configuration is required. In this paper, we propose to use the aspectual i approach to capture common and variable features in software product lines requirements. In doing so, we aim at addressing the issues pointed out previously and facilitating the selection among configuration alternatives to fulfill customer requirements.*

1. Introduction

Research in requirements for software product lines (SPLs) [5] has been exploring ways by which one can define core assets capable of serving as the basis for cost-effective derivation of products for individual users. Feature modeling [12] is a key technique for capturing commonalities and variabilities in system families and product lines. A feature may denote any functional or non-functional characteristic at the requirements, architecture, or any other level [6].

According to Czarnecki and Antkiewicz [7], although a feature model can represent commonalities and variabilities in a very concise taxonomic form, features in a feature model are merely symbols. Therefore, feature models lack foundations to reason about the relationships among different requirements [9] and artifacts of a variant [3]. They are not able to show how the features of an individual product fulfill the stakeholders' goals and, therefore, to keep trace between them. Moreover, feature models do not capture explicitly non-functional requirements and the positive/negative influence among them. This kind of reasoning would help to choose a specific configuration for an individual product according to the stakeholders' goals. The variability of the product line has to be documented explicitly to enable a strategic reuse of requirements artifacts. In this light, it seems clear that goal-oriented requirements engineering could be used to capture features using more meaningful models and, therefore, to keep trace of system features to their motivations.

Goal models provide a natural way to identify variability at the early requirements phase, by allowing the capture of alternative ways by which stakeholders achieve their goals [13, 15]. However, using only goal oriented approaches does not guarantee a proper capture of features variability. Indeed, a set of variable features differ from a specific product to another, i.e., they are not part of the common features of the product line. Besides, it is required an improved localization of features in software artifacts to facilitate the incremental evolution of feature functionality. Thus, these variable features could be modularized into aspects and later composed with common features in application engineering. Based on this, we present guidelines to map feature models to aspectual goal-oriented models. These models are defined based on the aspectual i* modeling language [1]. Aspectual goal-oriented model allows modeling stakeholders' goals, system requirements and aspects, and the relationships among them. We show how this model can be used to capture features in both domain engineering and application engineering. We argue that using an aspectual goal oriented approach can improve reuse and, therefore, reduce time and costs associated with evolving and configuring features to a specific product.

This paper is organized as follows. Section 2 overviews the aspectual i* modeling language and software product line engineering. Section 3 presents our mapping heuristics. Section 4 shows an example of using the proposed heuristics. Section 5 describes some related work. Finally, Section 6 summarizes our proposal and points out directions for future work.

[*] Currently in post-doc position at Universidade Nova de Lisboa

[ζ] This work was supported by several research grants: CAPES/GRICES Proc. 129/05; SOFTAS Project, POSC/EIA/60189/2004; European project AMPLE IST-033710.

2. Background

This section overviews the basis of our proposal, i.e., the aspectual i* modeling language and the principles of software product lines engineering.

2.1. Goal and Aspect Oriented Requirements Engineering

During the early stages of requirements engineering (RE), it is necessary to identify and specify how the intended system meets organizational goals, why the system is needed, what alternatives were considered, what the implications of the alternatives are for the various stakeholders, and how the stakeholders' interests and concerns might be addressed. The i* framework [17] is a goal-oriented approach widely adopted in the earlier phases of RE, as it offers a modeling language that describes the system and its environment in terms of actors and dependencies among them.

The i* framework offers two models: the Strategic Dependency (SD) and Strategic Rationale (SR). The SD model is described in terms of intentional relationships among strategic actors. An actor can depend upon another one to satisfy a goal, execute a task, provide a resource or satisfy a softgoal. Softgoals are associated to NFRs, while the other intentional elements are associated to system functionalities.

To illustrate some of the i*concepts and models, let us consider the Media Shop example [4]. Media Shop sells and ships different kinds of media items (e.g., books, newspapers, magazines, etc.). To increase market share, Media Shop has decided to open a Business-to-Consumer (B2C) retail sales front on the Internet (the Medi@ system). The goal is to allow an on-line customer to examine the items in its catalogue and place orders. For the sake of space, the SD model for this example is not shown in this paper, but an interested reader can find it in [4].

Figure 1 shows the SR model used to expand the description of the Medi@ actor. In this model three types of relationships are incorporated: (i) task-decomposition links describe what should be done to perform a certain task; (ii) means-end links suggest that one model element can be offered as a means to achieve another model element; (iii) contributions links suggest how a task can contribute to satisfy a softgoal (not shown in Figure 1).

Since the i* framework does not support the separation of crosscutting concerns, the approach presented in [1] has adopted principles of Aspect-Oriented Software Development (AOSD) [2] to identify candidate aspects in i* models, separate them in specific modules and compose them with other concerns.

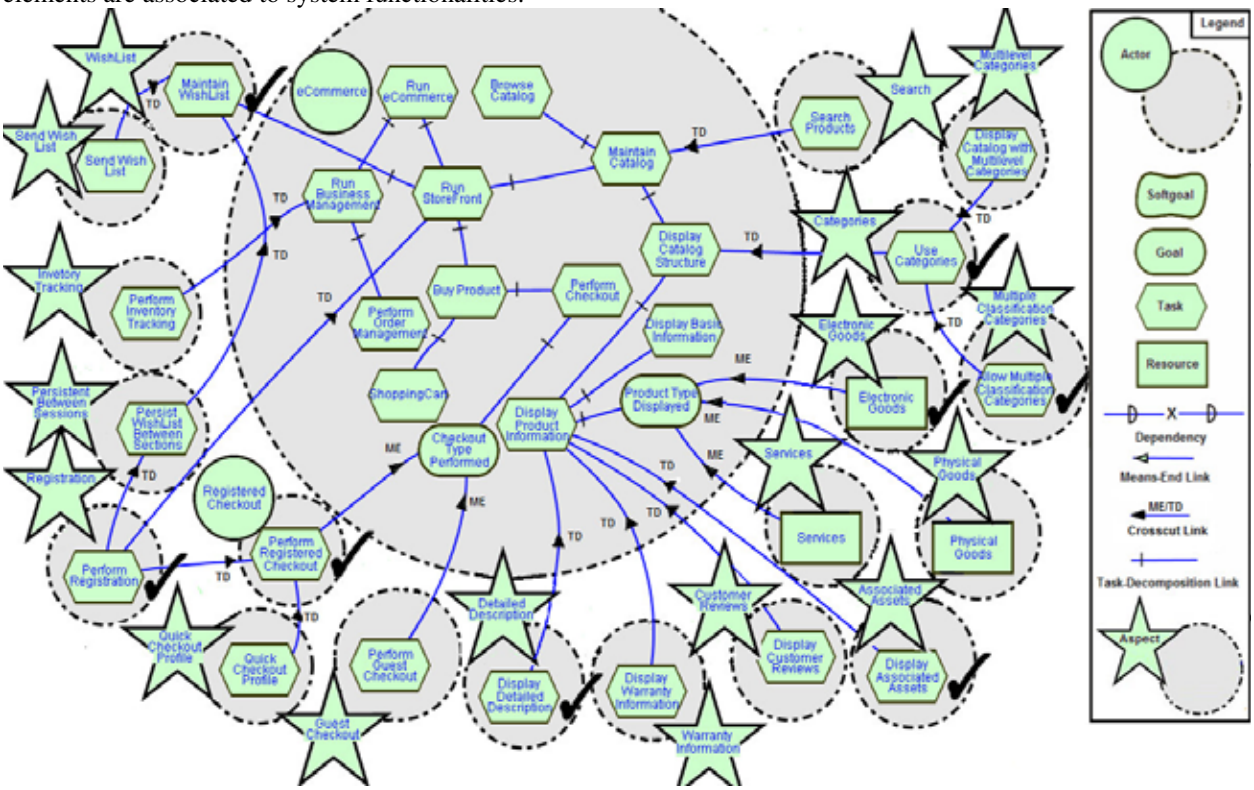


Figure 1. Aspectual i* model

For modularization purposes, and following the AOSD principles, we should externalize and modularize the identified crosscutting concerns, taking them away from the original actors, and place each of them in a new kind of model element, the aspect. The aspect is represented by a star (see Figure 1). A "crosscuts" relationship between the aspect and the actor (or another aspect) it affects is defined. The "crosscuts" relationship is represented by an arrow and its direction is indicated by a black triangle (see Figure 1) suggesting the composition direction. It means that the behavior of the source element needs to be transferred to the behavior of the target elements. The "crosscuts" relationships can be of two types: Means-End (ME) and Task-Decomposition (TD). These types are part of the composition rules defined to recover the relationships defined in the original model (e.g. the crosscuts TD link between Search aspect and Maintain Catalog task).

2.2. Software Product Lines Engineering

According to [5], a Software Product Line (SPL) is a set of software-intensive systems sharing a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way. Feature-oriented domain analysis gathers abstract concepts of the domain and organizes them as features [12]. A feature is a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate it among systems in a family. Feature modeling can be used at any stage of the software product-line engineering (e.g., requirements, architecture, design) and for any kind of artifacts (e.g., code, models, documentation). At an early stage, feature modeling enables product-line scoping to decide which features should be supported, or not, by a product line [6].

At its essence, a product line involves core asset development (also known as Domain Engineering) and product development using the core assets (also known as Application Engineering). Commonalities, as well as the flexibility to adapt to different product requirements are captured in core assets. Those reusable assets are created during domain engineering. During application engineering, products are either automatically or manually assembled, using the assets created during the domain engineering process and completed with product-specific artifacts. Thus, products differ by the set of features they include to fulfill customer requirements [16].

The feature model [6] describes the configuration space of a system family. An application engineer may specify a member of a system family by selecting the desired features from the feature model within the variability constraints defined by the model. Features can be mandatory, optional, or alternative.

To illustrate a feature model, let us consider a family of online B2C solutions presented in [7]. It is represented using the cardinality-based feature model [6] and a fragment of this model is shown in Figure 2. Feature cardinality is an interval denoting how often a feature with its subfeatures can be cloned as a child of its parent when specifying a concrete system. The model in Figure 2 contains one feature diagram, with eCommerce as its root feature.

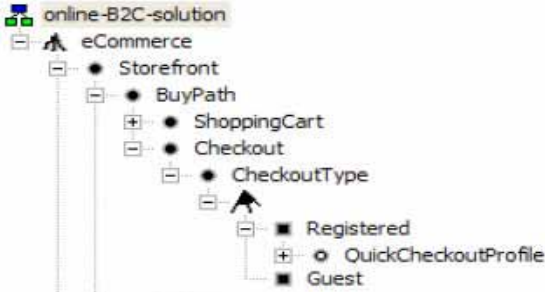


Figure 2. Online B2C Feature Model (taken from [7])

The root feature has a solitary subfeature: Storefront. The filled circle symbol indicates that Storefront has a feature cardinality of [1..1], meaning that the feature must exist once and only once. On the other hand, the empty circle symbol indicates that Quick Checkout Profile is an optional feature with cardinality [0..1]. Available checkout types, in this case Registered and Guest, are members of a feature group. The ramification symbol to denote a group indicates group cardinality (1– k), where k is the group size. Thus available checkout types can be any non-empty subset of the two checkout types. Grouped features are indicated by the filled square symbol.

In Section 3 we will illustrate how the aspectual i* framework can be used to both describe features in the domain engineering and in application engineering.

3. From Feature Models to Aspectual i* Models

Features are a de-facto standard in distinguishing the individual products in a product line, since each product is defined by a unique combination of features [12]. A feature may denote any functional or non-functional characteristic at the requirements level [6]. However, features in a feature model are merely symbols defining which features should be present in a product line and which should not. In fact, to specify in a requirements model "what is the system intended for?", a process view of the system to meet the business goals is required. Moreover, a justification of why a process is structured in a certain way, i.e., why a

certain alternative solution has been chosen, is also needed. In fact, mapping features to other models, such as the i* SR Model, can improve understanding the interdependencies among system features to achieve the expected system behavior.

Besides, aspect-oriented techniques have been applied to deal with highly volatile concerns [14] to increase software flexibility associated with the meeting of new requirements. Similarly, we can use the aspectual i* approach to modularize variabilities and to compose variable features with common features. In fact, optional and alternative features may be added or removed from a specific product, just as aspects. Representing these features as aspects can reduce time and costs associated with the configuration activity.

To illustrate how an aspectual goal oriented strategy can be used to represent variability in requirements models, we propose a set of heuristics to create aspectual i* models from feature models. These heuristics have been defined based on our previous experience [1, 14] and are described below:

H1. Separation of features in i* models:

(i) Mandatory features are part of the core assets of product lines and, therefore, can be mapped to internal elements in the actor representing the system;

(ii) Optional or Alternative features are part of the variable features of a product line and, therefore, can be mapped to internal aspect elements. In fact, some of them are going to be selected to configure the features of a specific product.

H2. Feature types and relationship among features in i* models:

(i) If a feature (mandatory or optional) is decomposed into other features, the root feature is mapped to a task and a task-decomposition link is used to relate the root feature with its sub-features: (a) If a sub-feature is not specialized into subtypes and is not a subtype of a super feature, then it is mapped to a task;

(ii) If a feature (mandatory or optional) is specialized into subtypes and they cannot be used simultaneously (i.e., they are alternative features), then a means-end link is used to relate the root feature with its subtypes and: (a) If the root feature denotes a functional requirement, it is mapped to a goal; (b) If the root feature denotes a non-functional requirement, it is mapped to a softgoal; (c) If the subtypes denote specialized ways of doing something, then they are mapped to tasks; (d) If the subtypes only denote types of information, then they are mapped to resources;

(iii) Relationships between features of type requires will be mapped to task-decomposition links from the required feature to the "requirer" feature. In this case, the "requirer" feature is a task.

H3. Check Relationship Correctness:

(i) If there is a relationship between an optional feature (encapsulated into an i* aspect) and a mandatory feature (encapsulated into an i* actor) or only between optional features, this relationship must be of type crosscuts: (a) If the relationship has been stated in previous guideline as task-decomposition link, then replace it by a crosscuts Task-Decomposition (TD) link; (b) If the relationship has been stated in previous guideline as means-end link, then replace it by a crosscuts Means-End (ME) link. Observe that often the names of the features could not be mapped directly to the names of internal elements in i*. Some adaptation needs to be made to produce a comprehensible i* models. The following heuristic is a step towards a systematic mapping of names.

H4. Mapping feature names to aspectual i* model element names:

(i) Mapping feature names to resources names: copy the name of the feature to the name of the resource;

(ii) Mapping feature names to task names: join the name of the feature with a verb to indicate an action in the present tense;

(iii) Mapping feature names to goals or softgoal names: join the name of the feature with a verb to indicate an action in the past tense;

(iv) Optional and alternative features (already mapped to aspects), have their names mapped directly to the names of the respective aspects. The biggest challenge in these mapping is that feature models describe the characteristics the system family have to present, but do not describe how these characteristics are achieved through system functionality and how these functionalities work together to achieve the expected system behavior. For this reason we cannot map feature models to aspectual i* models automatically. Indeed, it is needed to analyze the feature model to extract behavioral information to be used in the creation of aspectual i* models. To help this task, one can use the heuristics presented in previously.

4. Example: The Medi@ System

For the e-commerce system introduced in section 2.2, let us consider the feature model presented in [7] (whose fragment is shown in Figure 2) to apply the mapping heuristics and derive the aspectual i* model.

According to the mapping heuristics, the mandatory features are mapped to an internal element of the actor representing the commonalities of the product line. Notice that we add a meaningful verb to the name of a feature to denote functionality. For example, eCommerce, Buy Path, Shopping Cart, Checkout and Checkout Type are features mapped to the eCommerce actor in Figure 1 (H1-i). On the other hand, Registered,

Guest and Quick Checkout Profile are features that can be or not present in a specific product configuration. Thus, these features are modularized independently as aspects (H1-ii).

Decomposed features, such as Store Front, Buy Path and Checkout are all mapped into tasks (H2-i). Subfeatures, such as Persistent Between Sessions (not shown in Figure 2), are all mapped to tasks (H2-i-a). Relationships between decomposed features and optional sub-features must be stated as Crosscuts TD links (H3-i-a). For example, the relationship between Quick Checkout Profile and Registered. Specialized features, such as Checkout Type, are mapped into Goals since they denote system functionality (H2-ii-a). Feature subtypes, such as Registered and Guest, are mapped into tasks because they are ways of doing something (H2-ii-c), while Electronic Goods, Physical Goods and Services features (not shown in Figure 2) are mapped to resources because they are kinds of handled information (H2-ii-d). In the example found in [7] we could not find a case to use the heuristic "H2-ii-b". Relationships between specialized features and alternative sub-features must be stated as Crosscuts ME links (H3-i-b). For example, we have the relationship between Checkout Type and Registered.

After performing the domain engineering for the eCommerce example, we have captured both the common and the variable features of a family of online B2C solutions. The concrete system configured in the feature model presented in [7] is the same of the one presented in Figure 1. The next step is to select a set of features to configure a specific product to be developed in the application engineering.

A configuration in feature models specifies a concrete system and comprises the mandatory features plus the features checked with √ symbol (not shown in Figure 2 for the sake of space). In the configuration sample found in [7], checkout for registered customers is the only available checkout type, the catalog is sub-divided into categories, a product can be classified in multiple categories, the catalog contains only electronic goods, a wish list is maintained, etc.

Using the aspectual i* modeling language to capture the features in domain engineering also allows us to choose a configuration in application engineering. This makes easier building configuration models for a specific product and transferring the configuration information to other software artifacts (e.g. architectural design) in order to maintain tracing among different software artifacts of a variant. Besides, using a goal-oriented approach, such as i*, allows answering questions, such as "why a system configuration has been chosen?".

5. Related Work

The possibility of capturing variability in use cases is discussed in [11], where variation points are introduced within use case diagrams. Mapping feature models to UML 2.0 activity and class models have been proposed in [7]. The approach presented in [10] proposes concepts and tools that support the expression of feature-based variability in structural models and hence the selective adaptation of models. Goal orientation has been used in software customization approaches [13] where all variants are in one single system and the focus is on studying what customization is needed by a single user. Goals have also been used to explore alternatives [15] by focusing on exploring a space of alternatives before selecting the one to be implemented.

The approach presented in [8] uses goal oriented variability analysis to help selecting the best variant, given product requirements (functional and non-functional). In fact, NFRs (or softgoals) [17] provide natural rationale about why a given variant was selected. Thus, goal model brings to light the rationale behind variability by linking variants to softgoals.

In [9], the authors propose representing a requirements model with all possible combination of tasks (representing the system functionality) and softgoal solutions (variants) affecting them. This kind of combination will create a complex model. To reduce this complexity, they use aspect orientation to improve modularity. Thus, goal and softgoal graphs are maintained separately. In fact, they consider only softgoals as being aspects. Moreover, it is necessary to handle the complex interrelationships between goal and softgoal graphs that appear to analyze how softgoals solutions affect each part of the system (represented by tasks) and, therefore, producing a requirements model with variability. This reduction of relationships complexity is achieved by using labeling mechanism which attaches what softgoals solutions affect each task in the goal model. This approach considers variant as being alternative operationalizations of softgoals. We consider variant as alternative operationalizations of goals as well as additional functionalities (tasks), all of them modularized into aspects. It makes easier the configuration of new products by adding a set of variants to the system core assets through aspects weaving. Although our approach can also consider softgoals (NFRs) as being variants, we could not illustrate them in this paper because we have created an aspectual i* model from a feature model and the latter does not explicitly capture NFRs,

Most approaches define variability in terms of varying characteristics of the system-to-be, and not in

terms of the causes of these variations, i.e. the varying characteristics of the problem, the stakeholders and their needs. In [13], the authors propose a variability-intensive process for decomposing and analyzing goals. A key concept in this approach is the variability concerns, that is, types of questions whose alternative answers result in alternative refinements of the original goal. The collection of all concerns relevant to a goal is the variability frame evoked by the goal. Frames of variability concerns can be constructed from past projects, particularly artifacts of early elicitation efforts (e.g. interview transcripts, reports etc) and used in a concern-driven decomposition process. This process aims at attaining completeness in the variability acquisition results and allows reasoning about alternatives while taking into account the circumstances that hold in the context of attaining a goal. They capture variability in the early requirements phase before variation points of the system-to-be are defined. However, they are not concerned with the reuse of core assets or the easy configuration of specific products from domain engineering models. In fact, producing domain engineering models using aspect orientation makes it easier creating configuration models to be used in application engineering and improves the reuse of core assets.

6. Conclusions and Future Work

Our purpose in this paper was to show that the aspectual i* can be used to represent variability in domain and application engineering of SPL. In fact, we argue that the selection of specific features for an individual product and their composition with the core assets of SPL becomes facilitated due to the use of aspect orientation principles. Moreover, to enrich the variability captured by aspectual i*, we can combine it with approaches which elicit variability earlier in SPL development lifecycle, i.e., in the level of stakeholders' goals (e.g. [13]). In fact, variability in the stakeholders' goals will imply in variability in domain engineering. Also, capturing variability in aspectual i* can facilitate the interrelationships among several kinds of requirements (e.g., organizational, functional and non-functional) in the same model. This makes it easier answering why a specific feature configuration is required, and selecting among alternatives to fulfill customer requirements.

Future work includes investigating the integration of aspectual i* with the approach presented in [13]. Performing real case studies is also required to evaluate the widespread of the mapping heuristics. We also intend to investigate how features selected for a specific application and specified using the aspectual i* could be used to derive an aspect-oriented architectural design. Currently we are focusing on the improvement of models scalability by proposing structuring mechanisms such as views. The mapping heuristics cannot be performed automatically, since the mapping from feature models to aspectual i* models is not straightforward and some rationale is required from the analyst to perform this task.

References

[1] F. Alencar, et al. Integration of Aspects with i* Models, *Agent-Oriented Information Systems IV*, LNCS, Vol. 4898, Springer-Verlag, 2008, pp. 183-201.

[2] Aspect-Oriented Software Development (AOSD). Available at: http://www.aosd.net/. Last access in 01/08.

[3] S. Bühne, et al. Why is it not Sufficient to Model Requirements Variability with Feature Models?. *Ws on Automotive Requirements Eng*. IEEE Press, Japan. 2004

[4] J. Castro, et al. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems Journal*, 27(6), 2002, pp. 365-389.

[5] P. Clements, et al. *Software Product Lines: Practices and Patterns*, 1st ed. Boston, USA: Addison-Wesley. 2002.

[6] K. Czarnecki, et al. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, vol. 10, 2005, pp. 143-169.

[7] K. Czarnecki, et al. Mapping Features to Models: A Template Approach Based on Superimposed Variants. *4th Generative Programming and Component Engineering*, Estonia, Springer, 2005, pp. 422-437.

[8] B. González-Baixauli, et al. Visual Variability Analysis with Goal Models. *12th IEEE Requirements Engineering* (RE'04), IEEE Press, 2004.

[9] B. González-Baixauli, et al. Using Goal-Models to Analyze Variability. *1st Ws on Variability Modelling of Software-Intensive Systems*, Ireland, 2007, pp. 101-108.

[10] I. Groher and M. Völter. Expressing Feature-Based Variability in Structural Models. *10th Ws on Managing Variability for SPL at SPLC'07*, Kyoto, Japan, 2007.

[11] G. Halmans and K. Pohl. Communicating the variability of a software-product family to customers. *Software and System Modeling*, 2(1), 2003, pp. 15–36.

[12] K. Kang, et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study. *CMU/SEI-90-TR-021*, USA.

[13] S. Liaskos, et al. On Goal-based Variability Acquisition and Analysis. *14th IEEE Requirements Engineering* (RE'06), USA, 2006, pp. 76-85.

[14] A. Moreira, et al. Modeling Volatile Concerns as Aspects. *18th Conf. on Advanced Information Systems Engineering*, LNCS, 4001, Springer, 2006, pp. 544-558.

[15] J. Mylopoulos, et al. Exploring Alternatives during Requirements Analysis. *IEEE Software,* 18(1), 2001, pp. 92-96.

[16] K. Pohl, et al. *Software Product Line Engineering: Foundations, Principles, and Techniques*, Springer, 2005.

[17] E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. *3rd IEEE Requirements Engineering* (RE'97), USA, 1997.

REPRESENTING TEXTUAL REQUIREMENTS AS GRAPHICAL NATURAL LANGUAGE FOR UML DIAGRAM GENERATION

Magda G. Ilieva
Dept. of Computer Science and Software Engineering
Concordia University, Montreal, Canada
magda AT cse.concordia.ca

Harold Boley
Institute for Information Technology
National Research Council Canada, Fredericton, NB
harold.boley AT nrc.gc.ca

ABSTRACT
Since the establishment of the Unified Modeling Language (UML) as a standard graphical notation for representing knowledge, new ideas have emerged about tools that can automatically extract knowledge from text and represent it with UML diagrams. As the targeted representation of knowledge is in a graphical notation, we propose to also represent Natural Language (NL) and the knowledge it carries in a common graphical form, and then translate this Graphical NL (GNL) into another graphical form (UML).

KEY WORDS
Knowledge representation, Knowledge reformulation, NLP, Semantic Networks, UML, SE modelling

1. INTRODUCTION

Knowledge can be represented in variety of forms. In Software Engineering (SE), for example, perhaps the most common way of representing knowledge is with diagrams. This way of representation fits the understanding of a wide range of users. Graphical representation is done with self-explanatory shapes, it is semi-formal, and is suitable for subsequent formal processing into program code. This type of knowledge representation is easy to understand and widespread in information technology.

Quite often knowledge is extracted from text. Texts are written in Natural Language (NL), which is the universal method for representing knowledge. As the targeted model of knowledge extracted from text employs a graphical language, UML for example [12], why not also represent the source text itself graphically? We can then match the two graph models – UML and Graphical NL (GNL) [14] – and discover analogies as well as simplify translation. This article is organized as follows. After a review of related work, we explain the main principles of Graphical NL. Then, the use of GNL is demonstrated with a study case. In the fourth part analogies are presented between the two ways of graphical representation of knowledge – GNL and UML. These analogies are used for deriving rules for the automatic translation of textual user requirements into SE graph models. We conclude with an evaluation and comparison of the proposed GNL and other graphical representations for NL and knowledge.

2. RELATED WORK

The importance of automatic translation of software User Requirements (URs) from text to SE diagrams is evident from the continuing emergence of new theories and applications in this domain. In brief, the purpose of those applications is to automate user requirements analysis and to speed up the phase of software design.

Mainly, two types of expertise have to be united in order to develop technology for translating textual URs into SE diagrams: linguistic engineering and software engineering. Often, those two types of competence are applied in order to represent or reformulate knowledge during several stages. Reformulation consists of distinguishing and restructuring the initial natural language represented knowledge of humans in order to obtain formal language represented knowledge for computers. The first phase is Language Modeling (LM), which manages linguistic objects (texts, sentences, words, etc.) and their relations. The second phase, Knowledge Modeling (KM), defines concepts and relations, which are important for problem solving. Subsequent reformulation of knowledge, Intermediate Knowledge Modeling (IKM), is needed in order to obtain a form appropriate for mapping into a final SE model. Drawing those phases together (Fig1),we can use them as a frame for reviewing existing projects.

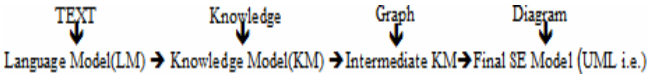


Fig1. Conceptual schema summarizing approaches

For example, in [7] LM is syntax patterns of restricted NL; KM consists of eight conceptual graphical patterns proposed for representing linguistic patterns extracted from text. Object Model (OM) and Behavior Model (BM), designed to capture the static and dynamic nature of requirements, serve as IKM (Intermediate Knowledge Model) from which the target OO diagram is derived. In a similar way, in [5], LM is restricted NL with particular syntax patterns; KM represents the types of data, operations over them and relations between them; for IKM a tree data structure is proposed, having three types of nodes: data, functionality and context. The authors in [6] consider KM as three types of graphs representing three types of knowledge for activity (emitted, absorbed and internal) extracted from restricted NL, namely, use case scenario specifications. Another example can be found in [4], where the few syntax constructs (LM) derived from the controlled language are grouped into relations (KM) that are subsequently represented as a conceptual lattice – an abstraction of a use case diagram. All of the above cited approaches obtain only one final graph model.

Other researchers focus on processing unrestricted NL. An example can be found in [1], where NL is modeled with a functional grammar, KM is presented as a Conceptual Prototyping Language, and two groups of graphs (IKM) are obtained – one for static knowledge and one for dynamic knowledge.

Another example can be found in [2, 3] where Case Grammar serves as LM while KM is presented as General Conceptual Model. Two IKMs are used for designing two target SE graph models: conceptual graphs - for obtaining activity diagrams and semantic networks - for supporting OO class diagrams.

A main point of correspondence between all theories is that they treat KM separately from LM. This separation limits the application of theories: they process specific types of knowledge applied to specific texts and receive one final graph model. Our approach differs here by offering a common graphical representation simultaneously of NL and the knowledge (both general and domain specific) included in it. After building the diagram of the text, we compare it with the diagram of the target SE graph model built by a human expert. Based on the discovered analogies between the two diagrams, we then define rules for translation of one graph into the other. This approach will make our methodology applicable to various texts, diverse knowledge and different target SE models. Our technology has fewer processing phases, which can increase its efficiency.

3. GRAPHICAL REPRESENTATION OF NL

Table structuring of an unrestricted NL: The graph representation of both language and knowledge in one unit is based on the graphical representation of relations between concepts. In order to represent text graphically we structure unrestricted NL into a table representation (TR). TR is described in [8,9], but for convenience we are going to discuss here one brief example from a case study: *"In a road traffic pricing system, drivers of authorized vehicles are charged at toll gates automatically. The gates are placed at special lanes called green lines. A driver has to install a device (a gizmo) in his/her vehicle."*

Structuring the text into a table is nothing but arranging it into three main columns – Su(bject), Pr(edicate) and Ob(ject), as also used in RDF. We obtain information for syntax structures and attached phrases when we process the text with one of the available POS taggers/chunkers [17]. Here is the outcome we got from the cited tagger:

1) In/IN ([a/DT road/NN traffic/NN]) pricing/VBG ([system/NN]),/, ([drivers/NNS]) of/IN ([authorized/JJ vehicle/NN]) **<: are/VBP charged/VBN :>**at/IN ([tool/NN gates/NNS]) automatically/RB./.

2) ([The/DT gates/NNS])**<:are/VBP placed/VBN:>**at/IN ([special/JJ lanes/NNS])**<: called/VBD :>**([green/JJ lines/NNS])./.

3) ([A/DT driver/NN])**<: has/VBZ to/TO install/VB :>**([a/DT device/NN]) (/(a/FW gizmo/FW)/) in/IN his/PRP$ //CC ([her/PRP$ vehicle/NN])./.

Tags for the syntax category of words, attached phrases and sets of rules are used in order to arrange the text into TR (shown in Tab 1). Su and Ob columns are noun phrases, Pr is a verb phrase.

Looking at TR, we can outline the following advantages:

i) TR is convenient for automatic processing: a) representation in another semi-formal notation, for example XML, and then, e.g., in SVG; b) fast access for storing and retrieving information; c) unlimited, expandable space with new rows for storing extra text and new columns for storing diverse syntactic and semantic information required for automatic text processing. We thus use TR as a knowledge base supporting text analysis.

Se	subse	Preconj	Su	Pr		Ob	Postconj
				verb	adverb		
1	1					In a road traffic pricing system	,
	2		drivers of authorized vehicles	are charged	automatically	at toll gates	.
2	1		The gates	are placed		at special lanes	
	2			called		green lines	.
3	1		A driver	has to install		a device (a gizmo) in his/her vehicle	.

Tab.1. Structuring text into a table

ii) The roles of phrases in sentences and relations between them are easy to explore. At the top-level of text structures we have a sequence of predicative relations.

iii) The relations in the next structural level are clearly distinguished – in each of the three components (Su, Pr, Ob). These relations can be summarized as: prepositional, noun-noun(s) modifier, adjective-noun modifier, verb-adverb.

iv) TR can be used as verification for the correctness of the tagging..

Basic building blocks of GNL: Table structuring helps us to reveal that NL can be represented graphically as ordered triplets (**concept1 relation concept2**). In order to define such a triplet we have to define its members:

Concepts are noun phrases which can be simple (consist of one noun) or complex (main noun with modifiers – adjective(s) or noun(s)). For example, *sensor* is a simple concept and *toll gate sensor* a complex one. Complex concepts consist of more than one noun, connected with a relation (implicit *has_a*). The interpretation of *"toll gate sensor"* is: *toll **has a** gate which **has a** sensor*.

Relation can be: predicative, prepositional, is_a, has_a.

. Predicative relation is defined as two concepts connected with a verb, for example: *A driver **installs** a device*;

. Prepositional relation is defined as two concepts connected with preposition. For example, *gizmo **in** vehicle*;

. Attributive ("noun *is* adjective" or "adjective noun"). For example, '*lane **is** green'* or '*green lane'*;

. Compositional relation could, in turn, fall into one of the following types:

- Noun-noun modifiers (*toll gate sensor*);
- Key-word/Enumerative structure (***types of** tool gate**:** single, entry, exit*); (*services**:** deposit, withdraw, transfer, get balance*).
- Possessive (*bank's client*);

In summary, all relations can be represented with a triplet, i.e. through Su, R, Ob. In a predicative relation R is a verb; in a prepositional relation R is a preposition; in an attributive relation R is equal to *is a*; in a compositional relation R is one of the following: *has a*, colon (:), key-words (*types of, kind of, consist of, include, …*).

Besides members, a relation has a direction.

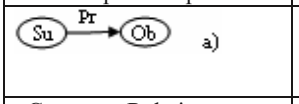
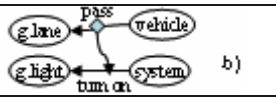
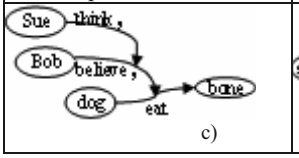
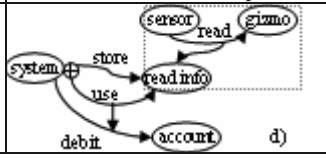
Direction signifies where the relation points. Predicative relations can have two directions: straight – from Su to Ob, which is represented through active voice, and reverse – from Ob to Su, represented through passive voice. In GNL

we represent a predicative relation through its straight direction, i.e. when we turn passive voice into active. The direction of a prepositional relation, too, does not match the order in which it is encountered in the prepositional phrase, which is from left to right, word after word. For example, "A to B" and "A from B" are two different directions. Or, the phrase "from A to B via C", does not mean to place them in order A,B,C but A,C,B. The attributive relations also have direction – for example, "green lanes" and "lanes are green" are the same relation represented with reversed directions. Opposite direction does not change the meaning of such relations, but it can change the importance of members when changing their positions. Normally, the most important member comes in the first position and becomes the head of the relation. This fact is used in some heuristics to discover empty positions in triplets (discussed in [14]).

During the process of restructuring the text into triplets some of the positions within a triplet may stay empty. Empty positions mean that their content is implicitly known from the context, or it is not important at this moment, or the sentence is not syntactically correct. For example, *the gate is placed at a special lane*, after changing the verb into active voice verb (straight direction), means that (Someone) (place) (the gate at a special lane). The position of Su (*Someone*) is left empty. It can remain empty until the analyst fills it or until we apply heuristics for discovering and filling it.

Graphical glue among triplets: In the previous section we discussed the decomposition of text into basic triplets. Their detailed graphical representation can be seen in [14]. In brief, a concept (noun) is represented as a solid oval; an attribute (adjective), as a dashed oval; a predicate, as a directed solid arc which connects related concepts; and a preposition, as a directed dashed arc which connects related concepts.

Now, we will explain how to graphically synthesize the diagram of an entire text from these basic triplets. In order to form a text representation, triplets are joined upon the relations between them. Relations are categorized upon the reason/result relationship between concepts/triplets. Tab 2 summarizes and gives examples of the different categories:

Concept-concept	Relation - Relation
	
Concept - Relation	Relation - Concept
	

Tab 2. Examples of relation types between triplets

The graph of a simple predicative relation, i.e. the ordered triplet Su, Pr, Ob, is represented as in Tab 2a). In Tab. 2b) a complex implicative relation between two relations is shown, representing the following text: *If a vehicle passes through a green line, the system turns on a green light*. Two simple predicative relations are connected into one complex, implicative relation via a directed arc connecting the predicative arcs of the simple sentences. At the start of the connecting arc there is a small diamond, which indicates the condition of an implication. Tab. 2c) shows an example (taken from [11]) of a simple (*eats*) relation at the end of a complex epistemic (*believes*) relation, which itself is at the end of another complex epistemic (*thinks*) relation*: Sue thinks that Bob believes that the dog eats a bone*. Three different relations are aggregated with a relative pronoun (*that*), which defines the direction and connections between them: *Sue thinks* ➔ *Bob believes* ➔ *the dog eats a bone*. Tab. 2d) shows an example of a resultant relation (framed as a box): *Sensor* ***reads*** *gizmo.* ***Read info*** *is stored by the system and used to debit account.* Both sentences have to be aggregated because the concept "*read info*" in the second sentence is the result of the activity "read" from the previous sentence. In the second sentence we have three simple predicative relations (*store, use* and *debit*) which form a complex sentence. We represent them as connected relations.

This was a summary of the principles which stand at the basis of graphic representation of text. The most important part of our methodology is to restructure the text in the form of basic triplets – relations, which would be subsequently represented in a unified graphic manner. In order to structure the text as basic triples we use technologies such as POS taggers, parsers, and chunkers. We write the basic building blocks (triplets: Su, Pr, Ob) into a table representation (TR), which helps us in further automated processing: i) turning the passive voice into active; ii) defining the heuristics and algorithms for filling out the empty positions of the triplets; iii) making it easier to resolve an anaphora and ellipsis. In [8,9] we described the stages of text analysis for the tabular representation of text. The Graphical Natural Language with which the text is made into a Semantic Network (SN) is described in [14]. Different aspects and applications of these TRs and SNs are described in [15].

4. CASE STUDY ON MODEL DISCOVERY

The objective of graphical NL is to represent concepts in a compact object-centered manner, i.e. to attach to each concept all relations in which it participates. This way we obtain a structured diagram of an entire text which shows the exact place and role of each concept, group of concepts, and connection between them. Fig. 2 illustrates a graphical representation of short text taken from [13]. Let us examine part of the diagram in order to explain how to read graphical symbols. We focus our attention on 'vehicle'.

Vehicle is a concept (noun) and as such it is represented inside an oval. *Vehicle* has two attributes – *authorized* and *non-authorized* (each attribute is represented inside a dashed oval). *Vehicle* participates in two predicative relations (drawn as solid lines) and three prepositional relations (dashed lines). The predicative relation that starts from *non-authorized vehicle* is labeled *pass*. It directs activity towards *green lane* and this activity is conditional (inside a diamond). If the condition is met, the implicative arrow that goes out of the diamond leads to one complex relation consisting of two simple relations connected conjunctively

(double circle): *System turns on yellow light* ***and*** *camera takes a photo*. *Photo* has a 'pin' with a number inside, which is compressed information about the photo (listed in a legend) and explains what *photo*'s role and features are. The graph which is obtained after processing the text has a lot of similarities with UML models. In order to show these similarities, we observe a part of the graphically represented text.

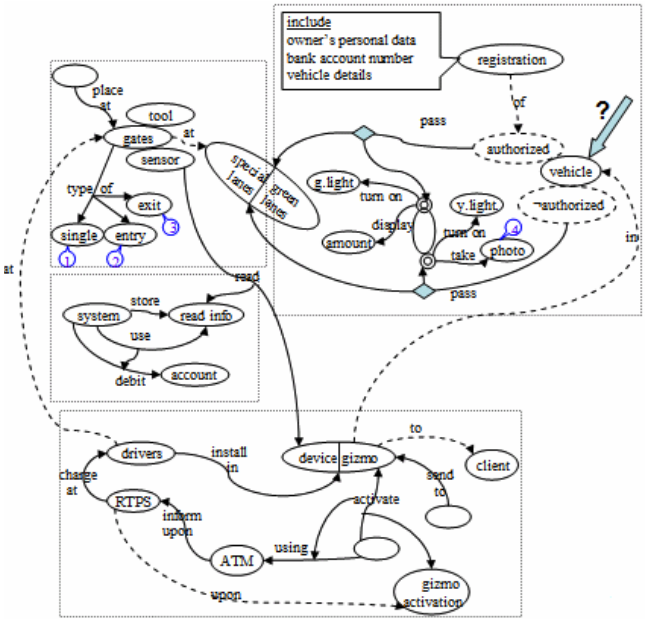


Fig.2. Graphical representation of text – Semantic Network

Domain model discovery: By a slight rearrangement of the shapes in a Semantic Network (SN) and ignoring the predicative relations, we notice that we can directly obtain a domain model (DM) from our GNL, as shown in Fig.3.

By analysis of the linguistic structure and DM structures, we come to defining the following rules for translation of SN into DM. Since DM is a static model and represents a hierarchical structuring of concepts, the following language structures are important for its generation: noun-noun attachment; adjective-noun attachment; prepositional attachment; key-word attachment. We use the term 'attachment' (rather than a phrase), to express the analogical relations that exist in NL and DM. We are interested in the static prepositions within prepositional attachments – the ones expressing place and possession. Key-word attachments are important for their representation of structural relations. For example: *consist of, involve, type of, part of, has a,* etc.

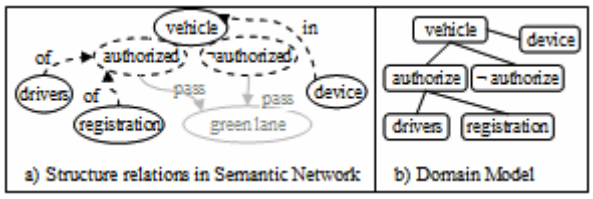


Fig.3. Mapping Semantic Network to Domain Model

Having defined which linguistic structures we have to translate into DM, we still have the knowledge engineering effort of the translation: extracting certain linguistic structures from the text, representing them in the nested format (see formula 4.1), defining operations over nested structures, simplifying, regrouping, and visualizing. The technology is described in detail in [16].

Object oriented (OO) model discovery: The concepts in the target model have properties and behavior. The first is a static characteristic while the second is dynamic. By analyzing our SN we notice that, apart from the structural relations (static), the nodes also have communicative relations (they 'send' and 'accept' predicative arcs). According to the number and type of predicative relations in which the different nodes of SN enter, they can be characterized as active and passive. The active nodes are candidates for objects in the OO model. By comparing in this way the peculiarities of the two graphic models – SN and OO diagrams, shown in Fig.4, we arrive at defining heuristics and rules for the translation of SN into OO diagrams. In general: **(i)** The domain model can serve as a structural basis for organizing the OO model. **(ii)** The nodes that are distinguished with attribute(s) / adjective(s) are candidates for parent nodes with instances. For example, instances of *vehicle* are *authorized vehicle* and *non-authorized vehicle*.

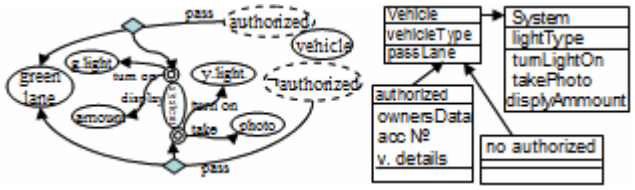


Fig.4. Mapping Semantic Network to Object Oriented Model

(iii) All predicative arcs which come out of "object nodes" are represented as methods. For example, *display* and *turn on* come from *system* and are represented as system methods; **(iv)** Terminal nodes – those that do not send predicative arcs – are regrouped as part of methods or data types. For example, *amount*, *photo*, *green light*, *yellow light*, are attached to the methods and represented as: displyAmount, takePhoto, turnLightOn. **(v)** Simplifying and regrouping, conceived for DM [16], can be applied to OOM. For example, two methods 'turn on *green* light' and 'turn on *yellow* light can be represented as one method with an argument thus:

turnOnLight (lightType (green, yellow)). (4.1)

We regroup the two adjectives of light into one abstract group, namely lightType; **(v)** the common methods of a node's instances are lifted to the parent node. For example, *passLane* is a method of *authorized* as well as of *non-authorized vehicle* and that's why we lift this method from instances to the parent node *Vehicle*. The same lifting technique is applicable for properties. The OO model is described in detail in [8].

Use Case Path (UCP) model discovery: Another type of model, which is important for the representation of the dynamics of a system, is derived from tracking different activities. The SN gives us a basis to arrange groups of concepts, as working nodes in which different actions are being executed. In our example from Fig.2 such structures of concepts (after their spatial arrangement guided by the prepositions for place with which they are connected) are as

shown in Fig.5: *vehicle* has *driver* and *gizmo*; *green lane* in which *toll gate* and *sensor* are placed; *RTP System*. If we write down the executable activities in the so-defined nodes, and we connect them with directed arcs in the order in which we read them in the text, we will obtain the diagram in Fig.5. In order to succeed in building this diagram, we change the point of view by considering triplets of the form 'actor-action-result'. We accept the following basic rule: the result of an activity is transferred, only if the working node is being changed, no matter if there is a recipient of the activity like it is in a UML sequence message chart. Based on this rule, no signal will go to *gizmo* after *install gizmo* or *activate gizmo*. *Driver* does not communicate with the other nodes. The type of *vehicle* is important for *System* to switch to *green or yellow light* and therefore verification of *vehicle type* is performed in the *System* node. The connection between *vehicle* and *system* is clear from the SN, while the connection between *vehicle* and *sensor* (depicted with a dashed line) has to be determined by the analyst.

The UCP model has no precise analog among the UML diagrams, but it is natural and stays close to the NL description of activities, hence can be used as an intermediary between NL and other UML diagrams (further explained in the article). The algorithm and a detailed description for processing this kind of diagram can be found in [15].

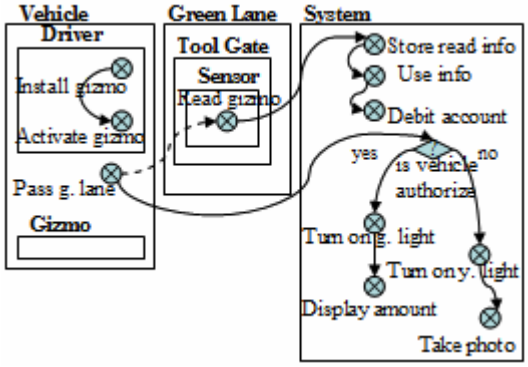


Fig.5. Use Case Path model

Hybrid Activity Diagram (HAD): This diagram can be obtained from UCP if we rearrange the working nodes as 'swimming lanes'. We inscribe the activities that are being executed in a swimming lane/working node in the same sequence in which we read them in the text. A message arrow connects swimming lanes in places where the result of the activity is being transferred. Following this logic we obtain the graph in Fig. 6a. Since our activity diagrams combine characteristics from both sequence message charts (swimming lanes and messages between them) and activity diagrams (conditional diamonds and activities), we call them Hybrid Activity Diagrams. From an **HAD** we can obtain sequence message charts by unrolling every path separately, as shown in Fig.6b).

Use Case (UC) model: In order to build this type of diagram, we are guided by the UML understanding of use cases as interactions only between the user and the system. The relations that we need from the text for this type of diagram are: i) only those in which the *user* is a Su, and the

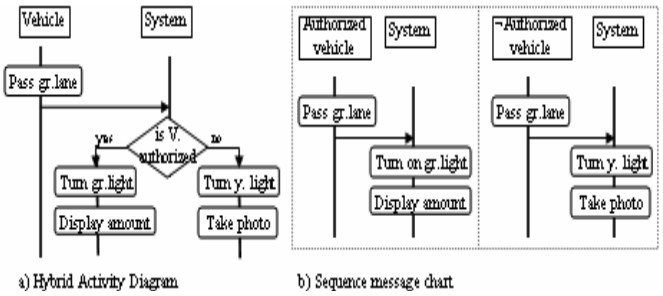


Fig.6. From an HAD to Sequence Message Charts

system is an Ob; ii) the *system* is a Su and continues an action initiated by the *user*. These types of relations, extracted from the graph of the text in Fig.2 are represented in Fig.7. The actions from case i) are connected with *user*, while those from case ii) are in the backend, and are represented as <extend> or <include>, depending on whether they are executed under specific condition, or not. For example, we observe in Fig.2., that *turn on green/yellow light* are activities placed after the diamond shape, i.e. they are conditional. In this case, activities will be included in the UC Diagram (UCD) as <extend> of the activity '*pass green lane'*. The activities of the system, with which a response is given to 'Sensor read gizmo', are not included into an UCD diagram, because there is no user participation, and thus they are not a part of the Use Case.

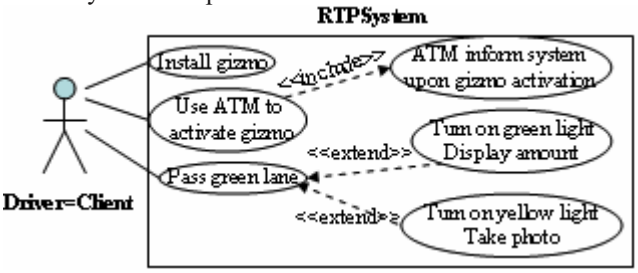


Fig.7. Use Case diagram

5. FINAL REMARKS

Summary: The idea of representing NL graphically is not new. Diverse graphical models keep appearing from both the fields of computer linguistics and SE modeling. While linguists tend to concentrate their efforts on the graphic representation of natural languages and aim to create more complete and precise models of languages, engineers are more interested in the domain knowledge, its extraction and representation. In order to automatically extract knowledge from text, we need a common model, which would represent both the text and the knowledge it contains. In order to make the model of the language (text) more universal and applicable to a wide range of problems, it has to represent both general and domain knowledge. While linguists offer models which mostly represent the general knowledge, engineers often prefer to create their own models of language, where they implicitly include specific domain knowledge. For example, the eight graphic templates proposed in [7] aim to summarize those characteristics of the NL model that are appropriate for its automatic mapping into an OO model. These templates are not likely to be appropriate for texts in which we cannot find these special language constructs, or for other target SE models. The

model in [5] is also obtained after the processing of special texts where the focus is on special linguistic templates, representing data structures and various processes applied to them.

The right balance between linguistic, general and problem domain knowledge in a single common representation has still not been discovered.

The current paper suggests one possible solution. We propose a unified model of natural language and the knowledge it carries. Working upward from the definition of natural language building blocks as relations between concepts, which are also building blocks of the knowledge represented by UML diagrams, we achieved a correlation between the two graphic representations. The graphic representation of text through Semantic Networks has served us in discovering patterns, analogical to the UML representation. This analogy helps us to reveal heuristics and rules with which the automatic generation of UML diagrams can be considered as a process of translating one graphic language into another.

Advantages of graphical formalisms: GNL was designed for SE purposes, namely, for creating executable models of knowledge described in natural language. GNL tries to capture unrestricted NL and to represent language and knowledge in one common model. That is what differentiates our methodology from other ones that separate the two models. The disadvantage of this separation is that if language patterns do not correspond to knowledge patterns, the theory loses validity.

In Fig.8 we present the same example in the two notations - Conceptual Graph formalism [10] and GNL. This brief visual comparison leads us to the following observations:

1) GNL is more compact, uses less space, and allows presenting larger volumes of information for visual inspection.

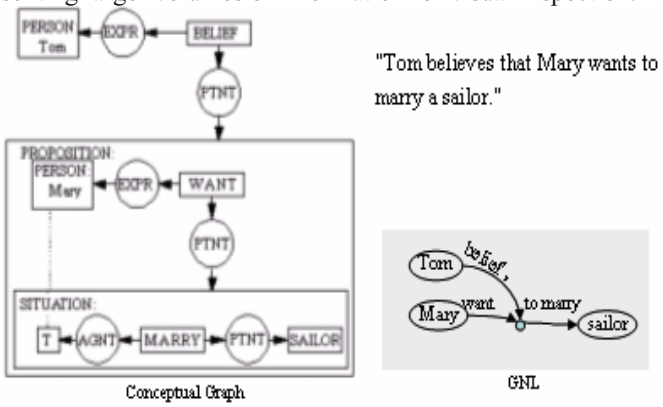


Fig.8. Two representations of the same example - comparison

2) In GNL, concepts and relations which form one simple sentence are free to participate in other relations too. This makes the concepts dynamic, and one concept can participate in many relations. **3)** As a consequence of the dynamism of the concepts in GNL, we can build a diagram of an entire text. **4)** The unambiguousness of the relations in GNL is supported by their strict indication with labels and with the use of different graphic symbols according to their semantic interpretation.

GNL is appropriate for the automatic drawing of text. An important supporting phase of its processing is the tabular representation (TR) of text. In order to construct a TR we use technologies of NL processing – POS taggers, parsers, chunkers. Then, for proceeding from TR to graphical and visual representations (e.g., SVG), it is possible to use scripting languages (e.g., PHP) and XML technologies.

Future work: We are going to develop GNL in two directions: **1)** Theoretical research which comprises the following: i) New extension to the knowledge base: examples, case studies, and comparison with examples from similar theories. ii) Add to, update and improve the collection of rules and heuristics. iii) Explore various methods and logical languages for the formal representation of SN. **2)** We will continue with the development of a software application which comprises the following projects: i) Architecture of an integrated environment for automatic analysis and formal representation of textual software requirements; ii) Structured representation of the text in a tabular format; iii) XML format of the TR; iv) Visualization.

REFERENCES

1. Burg, J.F.M. and van de Riet, R.P.: Analyzing Informal Requirements Specifications: A First Step towards Conceptual Modeling, Proc.of the 2nd Int. Workshop on Applications of Natural Language to Information Systems, Amsterdam, 1996.
2. Fliedl, G.; Kop, Ch.; Mayerthaler, W.; Mayr, H.C.; Winkler Ch.: The NIBA workflow: From textual requirements specifications to UML-schemata In: ICSSEA, Paris, 2002.
3. Kop, Ch.; Mayr, H.C.: Mapping Functional Requirements: From Natural Language to Conceptual Schemata, In Proc. of the 6th Int. Conf. SEA, Cambridge, USA, 2002.
4. D. Richards, K. Böttger, O. Aguilera: A Controlled Language to Assist Conversion of Use Case Descriptions into Concept Lattices. In 15th Australian Joint Conference on AI, 2002
5. Lee, B.-S., Bryant, B.R.: Automated conversion from requirements documentation to an object-oriented formal specification language. In Proceedings of SAC(ACM), Madrid, Spain, 2002.
6. Mencl, V.: Deriving Behavior Specifications from Textual Use Cases. In Proc of 'Workshop Intelligent Technologies for Software Engineering (WITSE, part of ASE), Linz, Austria, 2004.
7. Moreno A.: Object-Oriented Analysis from Textual Specifications", In Proc. of 9th International Conference on Software Engineering and Knowledge Engineering (SEKE), 1997.
8. Ilieva M., Ormandjieva O.: Automatic Transition of Natural Language Software Requirements Specification into Formal Representation, NLDB 2005.
9. M. G. Ilieva, O. Ormandjieva: Models Derived from Automatically Analyzed Textual User Requirements. Proc. of SERA'06
10. John F. Sowa: Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA, ©2000
11. John Sowa: SemNet http://www.jfsowa.com/pubs/semnet.htm
12. Unified Modeling Language (UML) 2.0 http://www.uml.org/
13. J. Araújo, A. Moreira, I. Brito, A. Rashid. Aspect-Oriented Requirements with UML. Workshop on "Aspect-oriented Modeling with UML", UML 2002, Dresden, Germany
14. Ilieva M.: Graphical Notation for Natural Language and Knowledge Representation. In Proc. of 19th SEKE, 2007.
15. Ilieva M.: Use Case Paths Model Revealing Through Natural Language Requirements Analysis, Proceedings of ICAI, 2007.
16. Ilieva M, Ormandjieva O.: NLP and FCA Technology for Automatic Building of DM, Proceedings of SEA, 2007
17. Infogistics' NLProcessor Interactive Demo: Tagging and Syntax Chunking http://www.infogistics.com/posdemo.htm

A Dynamic Adjusting Method for Test Case Prioritization*

Bo Qu Changhai Nie Baowen Xu Xiaofang Zhang
School of Computer Science and Engineering
Southeast University, Nanjing, China
{boqu, changhainie, bwxu, xfzhang}@seu.edu.cn

Abstract

Test case prioritization is an effective technique that helps to increase the rate of fault detection or code coverage in regression testing. In previous work, most of the techniques address on finding a best order of test cases to run. However, since the performance and context of program under test are unknown before the testing takes place, it is hard to build a best ordered test suite in advance. To address this problem, we propose a new dynamic adjusting method for test case prioritization. Our method uses feedback to sort test suit so that it can be gradually refined in regression testing phase. Compared with other techniques, our method has some merits in time complexity and applicable scenarios. The case studies also show that our new method is helpful in detecting more regression faults under some circumstances.

Keywords: Regression testing; dynamic adjusting; design information; test case prioritization;

1. Introduction

Regression testing is a necessary but expensive testing process as software evolves. To reduce the cost, numerous techniques have been reported in the literature on effective regression testing[1-11]. One effective approach for regression testing, test case prioritization, schedules test cases so that those with the higher priorities, according to their potential abilities to meeting some certain performance goals, are executed earlier in the regression testing process than lower priority test cases[2].

Most test case prioritization techniques focus on sorting and reusing test cases based on their historical performances[1,2,5,8]. To exhaust the use of the feedback, we present a new method that involves the use of the run time information. We firstly initialize the priorities of all test cases and an associated matrix which describes the relationship between test cases. Then we perform regression testing and consider feedback to adjust the priorities of unused test cases based on this matrix. The main contribution of our work is that we proposed a new dynamic adjusting method and its corresponding algorithm which has some merits in time complexity and applicable scenarios. We also perform a case study. The result shows that the new method is helpful in detecting faults in some circumstances.

2. Dynamic prioritization technique

2.1. General method

The definition of the test case prioritization problem suggests that prioritization techniques aim at finding a best prioritized test suite[8]. However, since the performance and context of program under test may be unknown before regression testing takes place, it is hard to build a best ordered test suite in advance. As an alternative method, we can initialize the test suite before testing. And then, after a test case is selected to run, we resort all unused test cases based on its feedback. So the test suite will be gradually refined.

A key point to this dynamic adjusting method is to find which test cases should be prioritized to higher or lower place. Consider a piece of code *P* and its modified version *P'* showed as follow:

```
if(b<0)          if(b<0)          t1: a=0, b=-1
  b = -b;          b = -b;        t2: a=1, b=1
c = b*a;         c = b*a;         t3: a=1, b=-1
                 d = b/a;         t4: a=0, b=1
```

Figure 1. Sample code and its modified version with test cases

Suppose there are four test cases which cover two set of execution paths, and t_1 and t_4 are designed to test boundary values. According to their historical fault detection numbers on *P*, the initial order of test cases is $t_1 > t_3 > t_2 > t_4$. When t_1 is selected and run on *P'*, if we set t_4 to higher priority due to their same designing goals, we could detect/locate the bug earlier.

*This work was supported in part by the National Natural Science Foundation of China (60425206, 60773104, 60633010), Excellent Talent Foundation on Teaching and Research of Southeast University, Doctor subject fund of education ministry (20060286020), and Jiangsu Planned Projects for Postdoctoral Research Funds (0601009B, 0701003B).

So we propose a dynamic adjusting method for test case prioritization with following steps: (1) Find test cases that have same or similar ability in detecting faults; (2) Re-evaluate the unused test case based on feedback; (3) Repeat step 2 until all test cases are selected to run. Section 2.2 will address step 1, and Section 2.3 will address step 2.

2.2. Similar relation and relation matrix

Test suite design information plays an important role in the testing phase. In test suite design information, testing objective of test case implicitly indicates the desired result. So test cases with same testing objective may reveal the same or similar regression faults. In order to describe the potential affiliation between different test cases which have the same testing objective, we here define the similar relation.

Definition 2.1: Given a test suite $T=\{t_1, t_2, \ldots, t_n\}$, test objectives $R=\{r_1, r_2, \ldots, r_n\}$, for each $t \in T$, let $f(t) \in R$ denotes the test objective covered by t. If t_i, $t_j \in T$ and $f(t_i)=f(t_j)$, we say t_i and t_j have a similar relation. This is written $t_i \sim t_j$.

Test cases with the same test objective may cover different codes or units. Therefore, they may not be redundant and should not be reduced. However, once a test case detected a defect, it indicates that the same or similar types of faults may exist elsewhere. So running other test cases with the same test objective earlier could be of benefit to detecting such unknown faults in time. On the other hand, once a test case passed, it enhances our confidences, so similar test cases could be set to lower priorities.

Definition 2.2: The relation matrix *MRS* is defined as a n×n matrix. This is written $MRS=(k_{i,j})_{n\times n}$, where element $k_{i,j}=1$ when $t_i \sim t_j$, otherwise, $k_{i,j}=0$.

MRS is defined to facilitate the using of such information between test cases. The algorithm of constructing *MRS* could be enumerating each test case and assigning the corresponding element value. Its time complexity would be no more than $O(n^2)$.

The advantages of using *MRS* are: (1) it could be easy to acquire information for constructing this matrix, and it is applicable for white or black box testing; (2) the cost of building *MRS* is low; (3) building and using this matrix would not involve collecting extra source related information, and it is independent from the size of target program.

2.3. Dynamic adjusting algorithm(DA)

In our prioritization method, we assign each test case a more flexible priority. During the regression testing procedure, its priority would not be a fixed value any more. Every value would be influenced by associated run-time results of test cases. So the test suite keeps evolving to adapt to the testing context gradually.

According to the testing result, we should deal with the failed or passed situation after one test case is executed. Here we define a structure $\Delta s=\{s1, s2\}$ ($s1,s2 \geq 0$), where *s1* denotes the adjusting range of associated test case's priority when test cases passed, and *s2* denotes that when test cases failed. The dynamic adjusting algorithm is showed in Algorithm 1.

```
     input      T': the selected test suite;  R: the relation matrix MRS; Δs: the range structure for test cases
     output     T'': the set of failed test cases
1.       for each t_m in T'
2.         if t_m failed then
3.           T'' += t_m
4.           for i = m+2 to n                          //adjust rest test cases. Ignore t_{m+1}.
5.             range = Min(i-m-1, Δs.s2)               //set max adjusting range
6.             if R_mi = 1 then                        //only adjust related test cases
7.               for j = i-1 to i-range step -1        //escalate priority in range
8.                 if R_mj = true then                 //see if there is other related test case in range
9.                   j = j + 1
10.                  break                             //ensure the order of test cases in one subset
11.              for k = i-1 to j step -1
12.                Swap t_k+1 and t_k                  //escalate the priority of ti
13.        else
14.          for i = n-1 to m+1 step -1                //adjust rest test cases. Ignore tn.
15.            range = Min(i-m-1, Δs.s1)               //set max adjusting range
16.            if R_mi = 1 then                        //only adjust related test cases
17.              for j = i+1 to i+range                //de-escalate priority in range
18.                if R_mj = true then                 //see if there is other related test case in range
19.                  j = j – 1
20.                  break                             //ensure the order of test cases in one subset
21.              for k = i +1 to j
22.                Swap t_k+1 and t_k                  //de-escalate the priority of ti
23.      return T''
```

Algorithm 1. Algorithm for dynamic adjusting priorities

2.4. Algorithm analysis

Compared to existing algorithms, our dynamic adjusting algorithm DA has some merits. For example, its time complexity is independent from the size of target program; it won't affect the order of subset in which test cases have the same testing objective.

Theorem 2.1: Time complexity of algorithm 1 is independent from size of target program.

Proof: Suppose the number of test cases is n, the size of program is m, the maximum adjusting range is s. The enumerating and running action takes place n times(line 2). Inside this loop, adjusting action takes place n times(line 5 and 20). Every single adjusting action would run $2s$ times, where s could be deemed as a constant. So the time complexity is $O(n^2)$. ■

Lemma 2.1: Algorithm 1 would not change the order of two test cases which have the similar relation.

Proof: Given test cases t_i and t_j, with $i<j$ and $t_i \sim t_j$, suppose test case t_k is executed.

When $k \geq i$, t_i has been executed, so the order of t_i and t_j would be affected.

When $k<i$ and t_k passes. If $MRS[k][i]=1$, that is $t_k \sim t_i$, since $t_i \sim t_j$, $MRS[k][j]=1$. Thus priorities of t_i and t_j are kept. If $MRS[k][i]=0$, because $t_i \sim t_j$, $MRS[k][j]=0$. Thus t_i would be at most exchange to the place of t_{j-1}. There is still $t_i<t_j$ after adjusting.

When $k<i$ and t_k fails. If $MRS[k][i]=0$, because $t_i \sim t_j$, $MRS[k][j]=0$. Thus priorities of t_i and t_j are kept. If $MRS[k][i]=1$, that is $t_k \sim t_i$, since $t_i \sim t_j$, $MRS[k][j]=1$. Thus t_j would be at most exchange to the place of t_{i+1}. There is still $t_i<t_j$ after adjusting.

To sum up, algorithm would not change the order of test cases t_i and t_j when $t_i \sim t_j$. ■

Theorem 2.2: Algorithm 1 would not affect the order of subset in which test case have the same testing objective.

Proof: For any $T'=\{t_{i1}, t_{i2}, t_{i3} \ldots t_{in}\}$ where T' is a subset of T and its all elements have the same testing objective. According to *Lemma* 3.1, for any test cases t_{im} and t_{in}, algorithm won't change the order. Thus, algorithm would not affect the order of T'. ■

3. Case study

3.1. Description

We choose Microsoft PowerPoint[1] to be our target program. We firstly randomly replace one or more bytes of normal PowerPoint (.ppt) documents, and then open the malicious documents with target program. If the document crashed or lead to a denied of service (DoS) of target programs, the test case is failed. Otherwise, the test case is passed.

According to experience, we have six different types of testing objectives, based on which we design and generate malicious documents as test cases. For example, if we want to detect null address reference, we randomly select four continuing bytes from normal document and set them to zero. Or if we want to detect the buffer overflow faults, we replace continuing bytes, often more than 255, to a null terminated string.

All test cases are generated and selected to execute in the former version of the target program. And we pick some of them which revealed faults in history to reuse. In our case study, test cases were selected and executed for 4 times. Each time, we use different Δs value. We assign that $\Delta s_1=\{0, 0\}$, $\Delta s_2=\{0, 2\}$, $\Delta s_3=\{1, 0\}$ and $\Delta s_4=\{1, 2\}$, where different Δs value represents different typical adjusting method.

3.2. Results and analysis

After running all 1656 test cases, there are 357 test cases failed, the rate of fault detection is about 21.65%. Table 1 shows the details.

Table 1. Failed and passed test cases

testing objectives	failed	passed	Total
Null address (NA)	142	267	409
Illegal pointer (IP)	34	212	246
Buffer overflow (BO)	9	209	218
Control exception (CE)	70	201	271
Illegal offset (IO)	40	257	297
Other exception (OE)	62	153	215

The time cost of executing all test cases is also recorded. Table 2 shows the time cost when different Δs value is used.

Table 2. Time cost

Δs	Δs_1	Δs_2	Δs_3	Δs_4
Time(sec)	34310	34991	35277	36017

According to Table 2, it is clear that when different Δs value is used, the time costs are nearly the same. Compared with no adjusting takes place(Δs_1), the dynamic adjusting strategy merely costs 3% more time on average(1.98%, 2.82% and 4.98% for Δs_2, Δs_3 and Δs_4 respectively). Since it is to wait program under test to generate results that spends more time in most testing situations, the extra cost brought by adjusting procedure is very low, and this won't obviously affect the total time cost.

Figure 3 shows the test results when different values of Δs are used. As observed in this figure, the performances of Δs_2, Δs_3 and Δs_4 are nearly the same, while the performance of Δs_2 is lower than others. The *APFD* value[7] for Δs_2, Δs_3 and Δs_4 are around 76%, and the *APFD* value for Δs_1 is about 70%. So in this

[1] "PowerPoint" is registered trademarks of Microsoft Corporation.

case, we can say the dynamic adjusting method(Δs_2, Δs_3 and Δs_4) achieves about 8.6% higher *APFD* value than that of no adjusting action performed(Δs_1).

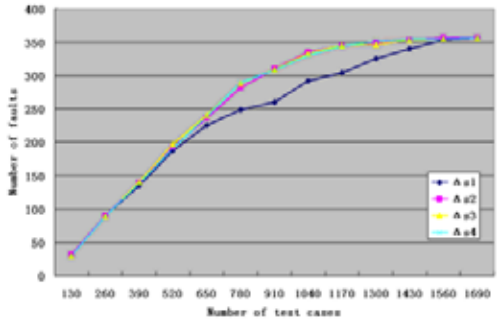


Figure 3. Test results

In practice, regression testing often stop before all test cases have been executed due to the budget. In such situations, more failed test cases are helpful for fault localization and correction. In such resource constraint case, if only 50% test cases could be executed, normal method(Δs_1) could detect about 70% faults, while the dynamic adjusting method(Δs_2, Δs_3 and Δs_4) could detect about 83% faults, which is about 20% higher. If the testing goal is to detect 80% regression faults, normal method(Δs_1) will spend about 21000 seconds. However, the dynamic adjusting method(Δs_2, Δs_3 and Δs_4) would spend only 15000 seconds to achieve this goal.

4. Conclusion and future work

The definition of the test case prioritization problem indicates that prioritization technique addresses on finding a best permutations of test cases to execute. Since the performance and context of program under test are unknown before regression testing takes place, it is hard to build a best ordered test suite in advance. Instead, we present a new dynamic adjusting method for test case prioritization based on test design information. The main idea behind our method is to gradually refine the test suite during the testing phase.

Though the theoretical analysis and the results of case study are encouraging, there are still many challenges ahead of us. First, the similar relation and its associated matrix approximately describe the potential relation of detecting regression faults between test cases, while this potential relation may be much more complex than what is depicted in a fixed matrix. So, an improved method could be building the matrix based upon test history and refining it during the testing phase. Second, different subset of test cases may correspond to different Δs value. This value could be given by experienced testers, or calculated in terms of test history rather than be assigned to a fixed number. Finally, we propose an algorithm that focuses on the order of each test case. However, the order of test cases is sometime represented by different selective probabilities. In the future, we are going to design new algorithms that adjust priorities in finer granularity. Also, we will carry out experiment to study if new algorithms would perform better.

References

[1] S. Elbaum, A. Malishevsky and G. Rothermel. Prioritizing test cases for regression testing. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 102-112, August 2000.

[2] S. Elbaum, A. Malishevsky and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 329-338, May 2001.

[3] S. Elbaum, A. G. Malishevsky and G. Rothermel. Test case prioritization: a family of empirical studies. *IEEE Transactions on Software Engineering*, 28(2):159–182, February 2002.

[4] S. Elbaum, G. Rothermel, S. Kanduri and A. G. Malishevsky. Selecting a cost-effective test case prioritization technique. *Software Quality Journal*, 12(3):185-210, September 2004.

[5] J. M. Kim and A. Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In *Proceedings of the 24th International Conference on Software Engineering*, pages 119-129, May 2002.

[6] Z. Li, M. Harman and R. M. Hierons. Search Algorithms for Regression Test Case Prioritization. *IEEE Transactions on Software Engineering*, 33(4):225-237, April 2007.

[7] G. Rothermel, R. H. Untch, C. Y. Chu, Mary J. Harrold. Test case prioritization: an empirical study. In *Proceedings of the International Conference on Software Maintenance*, September, 1999

[8] G. Rothermel, R. H. Untch, C. Y. Chu, M. J. Harrold. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, 27(10):929-948, October 2001.

[9] A. Srivastava and J. Thiagarajan. Effectively prioritizing tests in development environment. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 97-106, 2002.

[10] K. R. Walcott, M. L. Soffa, Gregory M. Kapfhammer and Robert S. Roos. Time-aware test suite prioritization. In *Proceedings of the International Symposium on Software Testing and Analysis*, page 1-12, July 2006.

[11] W. E. Wong, J. R. Horgan, S. London and H. Agrawal. A study of effective regression testing in practice. In *Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering*, pages 264-274, November 1997.

A Systematic Mapping Study on Non-Functional Search-Based Software Testing

Wasif Afzal, Richard Torkar and Robert Feldt
Blekinge Institute of Technology,
S-372 25 Ronneby, Sweden
{waf,rto,rfd}@bth.se

Abstract

Automated software test generation has been applied across the spectrum of test case design methods; this includes white-box (structural), black-box (functional), grey-box (combination of structural and functional) and non-functional testing. In this paper, we undertake a systematic mapping study to present a broad review of primary studies on the application of search-based optimization techniques to non-functional testing. The motivation is to identify the evidence available on the topic and to identify gaps in the application of search-based optimization techniques to different types of non-functional testing. The study is based on a comprehensive set of 35 papers obtained after using a multi-stage selection criteria and are published in workshops, conferences and journals in the time span 1996–2007. We conclude that the search-based software testing community needs to do more and broader studies on non-functional search-based software testing (NFSBST) and the results from our systematic map can help direct such efforts.

1. Introduction

Search-based software testing (SBST) research has attracted much attention in recent years as part of a general interest in search-based software engineering approaches [27, 28]. The growing interest in SBST can be attributed to the fact that there is a need for automatic generation of test data, since it is well-known that exhaustive testing is infeasible and the fact that software test data generation is considered NP-hard [36]. All approaches to SBST are based on satisfaction of a certain test adequacy criterion represented by a fitness function [27, 36]. McMinn [36] has written a survey on search-based software test generation, which shows the application of search-based techniques for white-box testing, black-box testing, grey-box testing and for the verification of non-functional properties. The survey shows that for non-functional testing, the search-based techniques are applied for execution time testing of real-time systems. Now, it is both important and interesting to know the extent of application of search-based optimization techniques for testing other non-functional properties. It is with this motivation that the current study has emerged from our work to gather, map and summarize primary studies about NFSBST in an accurate, fair and partial manner [34]. It is essentially a systematic mapping study to identify available evidence on NFSBST. A *systematic map* provides an overview of a research area to assess the quantity of evidence existing on a topic of interest [34] (see e.g. Bailey's et al. mapping study [4]).

The remainder of this paper is organized as follows. Section 2 describes our research protocol, including the search strategy and study selection. In Section 3, we describe the results. Sections 4 and 5 comprises of analysis and discussion of results, while the paper is concluded in Section 6.

2. Identification of research

We defined the following research question inline with the overall purpose of the study:

RQ: In which non-functional testing areas have search-based techniques been applied and what are the different metaheuristics used ?

A clear definition of population, intervention, outcomes and experimental design helps identifying relevant primary studies [34]. Our population is limited to the application area of software testing. Our intervention includes application of metaheuristic search techniques to test different types of non-functional properties. The outcome of our interest represents different types of non-functional testing that use metaheuristic search techniques.

2.1. Generating a search strategy

We used the following search terms to find relevant papers:

- **Population:** testing, software testing, testing software, test data generation, automated testing, automatic testing.

- **Intervention:** evolutionary, heuristic, search-based, metaheuristic, optimization, hill-climbing, simulated annealing, tabu search, genetic algorithms, genetic programming.

- **Outcomes:** non-functional, safety, robustness, stress, security, usability, integrity, efficiency, reliability, maintainability, testability, flexibility, reusability, portability, interoperability, performance, availability, scalability

We used Boolean `OR` to join alternate words and synonyms and Boolean `AND` to join major terms for population, intervention and outcome. The non-functional properties listed under outcomes are guided by five existing taxonomies, namely McCall software quality model, Boehm software quality model [19], ISO/IEC 9126-1 [30], IEEE Standard 830-1998 [29] and Donald G. Firesmith's taxonomy [20]. The non-functional properties obtained from existing taxonomies are restricted to high-level external attributes only for the sole purpose of guiding the search strategy. The different non-functional testing areas that are discussed later in the paper cannot be mapped as it is with these listed non-functional properties. Therefore, while quality of service includes attributes such as availability and reliability, we have retained the term quality of service in later part of the paper (Subsection 4.2) to remain consistent with the terms used by the original authors in their respective papers. Similarly, one can argue execution time (Subsection 4.1) and buffer overflow (Subsection 4.3) to fit under performance and security respectively, but we remain consistent with using the common terms of execution time and buffer overflow according to the authors' usage.

The search was applied on digital libraries accessed via IEEE Xplore, ACM Digital Library, Compendex and ISI Web of Science. In addition, manual search was performed on the following journals (J) and conference proceedings (C): Real Time Systems Symposium (C), Real Time Systems (J), Genetic and Evolutionary Computation Conference—Search-based Software Engineering Track (C),Software Testing, Verification and Reliability (J) and Software Quality Journal (J). To have confidence in the completeness of search, the results of the search were matched against a core set of studies to compare that the search found the entire core set.

To have a more representative set of studies, we also scanned the reference lists of primary studies and contacted researchers who authored most of the papers in a particular non-functional area. Only studies within the time span 1996–2007 were included. It is important to note that having restricted the search within these years excluded studies by Schultz et al. [42, 43] (authored in year 1992 and 1995 respectively) which applies evolutionary algorithms for robustness testing of autonomous vehicle controllers. We therefore, do not include these two studies in the analysis.

2.2. Study selection

Optimization techniques have been applied across different engineering and scientific disciplines. Moreover within software testing, search techniques have been applied from planning to execution. Therefore, it is imperative that we define comprehensive inclusion/exclusion criteria. We excluded studies that do not relate to software engineering/development, do not relate to software testing, do not report application of optimization techniques, do not report application of metaheuristics (metaheuristics include hill climbing, simulated annealing, tabu search, ant colony methods, swarm intelligence and evolutionary methods [10]), describe search-based testing approaches which are inherently structural (white-box), functional (black-box) or grey-box (combination of structural and functional) (this exclusion criterion is relaxed to include those studies where a structural test criterion is used to test non-functional properties, e.g. [5]), are not related to the testing of the end product e.g. [55], are related to test planning e.g. [16], make use of model checking and formal methods e.g. [3, 17], report performance of a particular metaheuristic instead of its application to software testing e.g. [35], report on test case prioritization e.g. [50], are used for prediction and estimation of software properties e.g. [6, 44].

In the beginning, a single researcher excluded 37 references out of a total of 404, primarily based on reading the title and abstract. The remaining 367 references were subjected to detailed exclusion criteria, which involved three researchers. This resulted in 60 remaining papers, which were further filtered out by reading full-text. A final figure of 24 primary studies was reached after excluding similar studies that were published in different venues. The 24 primary studies were complemented with 11 more papers by scanning the reference lists of the primary studies and contacting relevant authors.

3. Results

The results indicate that within non-functional testing, the application of metaheuristic search techniques can be classified under execution time, quality of service (QoS), buffer overflow, usability, and safety.

Figure 1 shows the year-wise distribution of primary studies within each non-functional property as well as the frequency of application of different metaheuristics. The

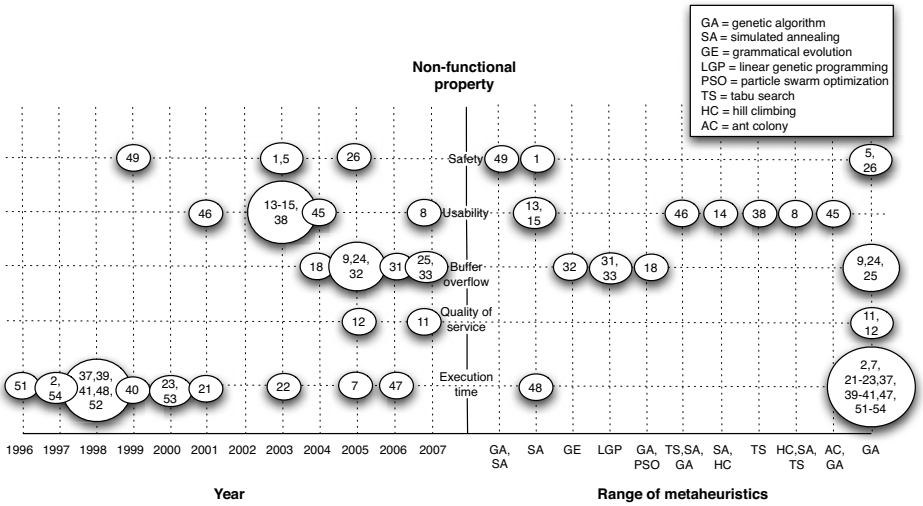


Figure 1. Distribution of NFSBST research over range of applied metaheuristics and time period.

bubble at the intersection of axes contains the reference number of papers. It is evident from the figure that genetic algorithms are the most widely used metaheuristic with applications in 21 papers across different types of non-functional testing. In the left quadrant of Figure 1, each bubble represents the reference numbers of primary studies within each non-functional area in respective years from 1996–2007.

4. Analysis

The focus of this section is to present a broad overview of research within NFSBST, discussion of range of metaheuristic techniques used and satisfaction of problem objectives.

4.1. Execution time

The application of metaheuristic search techniques to test real-time requirements in embedded computer systems involves finding the best and worst case execution times (BCET, WCET) to determine if timing constraints are fulfilled. Our systematic map indicates that the papers measuring BCET and WCET are by far the largest contributor in NFSBST research. The study by Briand et al. [7] has differentiated the temporal testing research into two directions. The one direction focuses on violation of timing constraints due to input values and has attracted the bulk of research. The other direction, which is the one taken by Briand et al. [7], analyses task architectures and consider seeding times of events triggering tasks and tasks' synchronization. This study does not considers tasks in isolation. Both approaches to temporal verification, however, are complementary. Another dimension of research into temporal testing using metaheuristic search techniques focuses on properties of test objects inhibiting evolutionary testability and formulation of complexity measures for predicting evolutionary testability [21, 22].

Genetic algorithms have been used as the metaheuristic in majority of studies (14 out of 15). The fitness functions vary according to research dimensions described above, which includes measurement of execution time of the test object, coverage of code annotations inserted along shortest and longest algorithmic execution paths and exponential fitness function based on the difference between execution's deadline and execution's actual completion.

4.2. Quality of Service (QoS)

Under the umbrella of service-oriented software engineering, genetic algorithms have been used for quality of service aware composition and violation of service level agreements (SLAs) between the integrator and the end user. The range of fitness functions used are based on the maximization of desired QoS attributes with a static or dynamic penalty function and a combination of distance-based fitness with a fitness guiding the coverage of target statements.

4.3. Buffer overflow

Buffer overflow can cause unauthorized exploits, thus compromising software security. Grammatical evolution, linear genetic programming, genetic algorithm and particle swarm optimization have been used for detecting buffer

overflows. The objective is to detect buffer overflows, vulnerable statements, exceptions and evolving plausible attacks. Most of the fitness functions are based on the ability of an attack to fulfill the conditions necessary for a successful exploit. The work of Kayacik et al. [31, 32, 33] is notable as they describe an approach to a framework for attack generation based on the evolution of system call sequences.

4.4. Usability

Search-based usability testing of software has been applied in the form of interaction testing where the goal is to test the t-way interactions taking place through the user interface. The research into interaction testing has focused on generating covering arrays which is a combinatorial object representing interactions. These studies show the use of hill-climbing, simulated annealing, tabu search, genetic algorithms and ant colony algorithms as the applied metaheuristics. The objective is either rapid coverage of interactions or obtaining smaller test suites. The fitness function used for constructing covering array is the number of uncovered t-subsets.

4.5. Safety

Search-based safety testing is an area where the research has targeted real world problems such as safety of car control systems [5] and steam boilers [1]. The research into search-based safety testing can be differentiated into two themes. One is the case where generation of separate inputs is discussed to test the safety property while the other case discusses generation of sequence of inputs. The objective is the violation of a safety property. The used metaheuristics include genetic algorithms and simulated annealing. The fitness functions used measures the cost related to the violation of the safety property.

5. Discussion

We presented the results of the initial scoping study (systematic map) to identify the extent and form of literature within NFSBST. The results of our systematic map indicates that NFSBST is focused on five areas, with execution time testing being the most researched non-functional property. This indicates that execution time testing represents a suitable search problem. On the other hand, for execution time testing this might also mark the beginning of more in-depth analysis of problem characteristics including comparative and performance evaluation studies [21, 22]. As compared to execution time testing, the application of metaheuristic search techniques for detecting buffer overflows, usability testing, safety testing and quality of service is more recent. Further feedback from empirical studies into these niche non-functional areas is required to gain confidence into the efficacy when applying search-based techniques.

We also find that the current taxonomies for non-functional properties need to assemble a more complete set of non-functional properties for software systems.

Apart from the final set of 35 papers, our search also resulted in studies which, although, applies search-based techniques, are not related to test data generation. Examples of such studies include reliability modeling [44] and test planning [16]. Studies relevant to test planning reflects the growing application of metaheuristics across the software testing lifecycle, while studies related to reliability modeling offers yet another dimension where the application of search techniques can offer near optimal solutions. These studies, together with existing SBST literature, can offer an exciting future arena where studies are not only limited to automated software test data generation but also extended to address broader verification and validation problems that are open to the application of search-based techniques. Our future work with a systematic review should explore these possibilities in more detail.

In terms of validity threats, there is a possibility that we might have missed relevant studies. However, our rigorous search strategy (Subsection 2.1) should have assembled a reasonable sample.

6. Conclusions

This work presents initial findings related to the application of metaheuristic search techniques to test non-functional properties. A total of 35 papers published in the years 1996–2007 are used a basis to map the application of metaheuristic search techniques to five different non-functional areas of execution time, quality of service, buffer overflow, usability and safety.

We presented an analysis of these studies in terms of problem objective, applied metaheuristic and range of fitness functions used. A large percentage (42.8%) of the studies deal with execution time testing with evidence of experimentation with real world applications. Regarding the rest of the non-functional properties, further feedback from empirical studies is desirable. We also found that diverse metaheuristic search techniques have been applied to achieve problem-specific objectives, with genetic algorithms being the most frequently used metaheuristic.

There is still plenty of potential for automating non-functional testing using search-based techniques. The results of our systematic map also indicate that the current body of knowledge concerning SBST does not report studies on many of the other non-functional properties. On the other hand, there is a need to extend the early optimistic results of applying NFSBST to larger real world systems, thus

moving towards a generalization of results.

Future work includes extending the presented results into a systematic literature review.

References

[1] O. Abdellatif-Kaddour, P. Thevenod-Fosse, and H. Waeselynck. Property-Oriented Testing based on Simulated Annealing. In *Proceedings of ACS/IEEE International Conference on Computer Systems and Applications (AICCSA' 2003)*, Tunis (Tunisie), 2003.

[2] J. T. Alander, T. Mantere, and G. Moghadampour. Searching Protection Relay Response Time Extremes using Genetic Algorithm – Software Quality by Optimization. In *Proceedings of the 4th International Conference on Advances in Power System Control, Operation and Management, APSCOM-97*, Hong Kong, November 1997.

[3] E. Alba and F. Chicano. Finding Safety Errors with ACO. In *Proceedings of 9th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, USA, 2007.

[4] J. Bailey, D. Budgen, M. Turner, B. Kitchenham, P. Brereton, and S. Linkman. Evidence Relating to Object-Oriented Software Design: A Survey. In *Proceedings of First International Symposium on Empirical Software Engineering and Measurement*, IEEE, 2007.

[5] A. Baresel, H. Pohlheim, and S. Sadeghipour. Structural and Functional Sequence Test of Dynamic and State-Based Software with Evolutionary Algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2003), Lecture Notes in Computer Science (LNCS 2724)*, Springer–Verlag, Berlin, Germany, 2003.

[6] S. Boukif, H. Sahraoui, and G. Antoniol. Simulated Annealing for Software Quality Prediction. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, USA, 2006.

[7] L. C. Briand, Y. Labiche, and M. Shousha. Stress Testing Real-Time Systems with Genetic Algorithms. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, GECCO 05*, Washington, DC, USA, June 25–29 2005.

[8] R. C. Bryce and C. J. Colbourn. One-Test-at-a-Time Heuristic Search for Interaction Test Suits. In *Proceedings of the 2007 Conference on Genetic and Evolutionary Computation, GECCO 07*, London, UK, July 7–11, 2007.

[9] J. Budynek, E. Bonabeau, and B. Shargel. Evolving Computer Intrusion Scripts for Vulnerability Assessment and Log Analysis. In *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, GECCO 05*, pages 153–160, Washington, DC, USA, June 25–29, 2005.

[10] E. K. Burke and G. Kendall. *Search Methodologies – Introductory Tutorials in Optimization and Decision Support Techniques*. Springer Science and Business Media, New York, USA, 2005.

[11] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. Search-based Testing of Service Level Agreements. In *Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO '07*, London, UK, July 7–11, 2007.

[12] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An Approach for QoS-aware Service Composition based on Genetic Algorithms. In *GECCO 05: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, Washington, DC, USA, June 25–29, 2005.

[13] M. B. Cohen, C. J. Colbourn, and A. C. H. Ling. Constructing Strength Three Covering Arrays with Augmented Annealing. *Discrete Mathematics*, 2003.

[14] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, and C. J. Colbourn. Constructing Test Suites for Interaction Testing. In *Proceedings of the 25th International Conference on Software Engineering (ICSE 03)*, IEEE, 2003.

[15] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, C. J. Colbourn, and J. S. Collofello. Variable Strength Interaction Testing of Components. In *Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC 03)*, IEEE, 2003.

[16] Y. S. Dal, M. Xie, K. L. Poh, and B. Yang. Optimal Testing-Resource Allocation with Genetic Algorithm for Modular Software Systems. *The Journal of Systems and Software*, 66(1), 2003.

[17] K. Derdarian, R. M. Hierons, M. Harman, and Q. Guo. Input Sequence Generation for Testing of Communicating Finite State Machines (CFSMs). In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2004), Lecture Notes in Computer Science (LNCS 3103), Springer–Verlag*, Berlin, Germany, 2004.

[18] G. Dozier, D. Brown, J. Hurley, and K. Cain. Vulnerability Analysis of Immunity-Based Intrusion Detection Systems Using Evolutionary Hackers. In *Proceedings of the 2004 Conference on Genetic and Evolutionary Computation, GECCO 04*, Seattle, Washington, USA, June 26–30, 2004.

[19] N. E. Fenton and S. L. Pfleeger. *Sofware Metrics - A Rigorous and Practical Approach, 2nd Edition*. International Thomson Computer Press, Boston, USA, 1996.

[20] D. G. Firesmith. Common Concepts Underlying Safety, Security, and Survivability Engineering. Technical Note CMU/SEI-2003-TN-033, Carnegie Mellon Software Engineering Institute, 2003.

[21] H. G. Gross. A Prediction System for Dynamic Optimization-based Execution Time Analysis. In *Proceedings of First International Workshop on Software Engineering using Metaheuristic Innovative Algorithms (SEMINAL), ICSE 2001*, Toronto, 2001.

[22] H. G. Gross. An Evaluation of Dynamic, Optimization-based Worst-case Execution Time Analysis. In *Proceedings of the International Conference on Information Technology: Prospects and Challenges in the 21st Century*, Kathmandu, Nepal, 2003.

[23] H. G. Gross, B. F. Jones, and D. E. Eyres. Structural Performance Measure of Evolutionary Testing Applied to Worst-case Timing of Real-time Systems. *Proceedings of IEE Software*, 147(2), 2000.

[24] C. Grosso, G. Antoniol, M. D. Penta, P. Galinier, and E. Merlo. Improving Network Applications Security: a New Heuristic to Generate Stress Testing Data. In *Proceedings of the Annual Conference on Genetic and Evolutionary Computation, GECCO 05*, ACM, New York, USA, 2005.

[25] C. D. Grosso, G. Antoniol, E. Merlo, and P. Galinear. Detecting Buffer Overow via Automatic Test Input Data Generation. *Computers and Operations Research, Elsevier*, 2007.

[26] A. G. H. Pohlheim, M. Conrad. Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences. In *SAE 2005 World Congress Exhibition*, Detroit, MI, USA, April 2005.

[27] M. Harman. The Current State and Future of Search-based Software Engineering. In *Proceedings of Future of Software Engineering (FOSE 07) at 29th International Conference on Software Engineering*. IEEE Computer Society, USA, 2007.

[28] M. Harman and B. Jones. Search-based Software Engineering. *Information and Software Technology*, 43(14), 2001.

[29] IEEE Std 830-1998. *IEEE Recommended Practice for Software Requirements Specifications*, 1998.

[30] International Standard. *ISO/IEC 9126-1:2001 Software Engineering Product Quality Part 1: Quality Model*, 2001.

[31] H. G. Kayacik, M. Heywood, and A. N. Zincir-Heywood. On Evolving Buffer Overflow Attacks Using Genetic Programming. In *Proceedings of the 2006 Conference on Genetic and Evolutionary Computation, GECCO 06*, Seattle, Washington, USA, July 8–12, 2006.

[32] H. G. Kayacik, A. N. Zincir-Heywood, and M. Heywood. Evolving Successful Stack Overflow Attacks for Vulnerability Testing. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC 2005)*, IEEE, 2005.

[33] H. G. Kayacik, A. N. Zincir-Heywood, and M. Heywood. Automatically Evading IDS Using GP Authored Attacks. In *Proceedings of the IEEE Computational intelligence in Security and Defense Applications - CISDA 2007*, pages 153–160, April 2007.

[34] B. Kitchenham. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE-2007-01, Keele University and University of Durham, UK, 2007.

[35] J. Koljonen, M. Mannila, and M. Wanne. Testing the Performance of a 2D Nearest Point Algorithm with Genetic Algorithm Generated Gaussian Distributions. *Expert Systems with Applications*, 32(3), 2007.

[36] P. McMinn. Search-Based Software Test Data Generation: A Survey. *Software Testing,Verification and Reliability*, 14(2), 2004.

[37] F. Mueller and J. Wegener. A Comparison of Static Analysis and Evolutionary Testing for the Verication of Timing Constraints. In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium*, Denver, USA, June 1998.

[38] K. J. Nurmela. Upper Bounds for Covering Arrays by Tabu Search. *Discrete Applied Mathematics, Elsevier*, 2003.

[39] M. OSullivan, S. Vossner, and J. Wegener. Testing Temporal Correctness of Real-Time Systems - A New Approach using Genetic Algorithms and Cluster Analysis. In *Proceedings of the 6th European Conference on Software Testing, Analysis Review (EuroSTAR 1998)*, Munich, Germany, 1998.

[40] H. Pohlheim and J. Wegener. Testing the Temporal Behavior of Real-Time Software Modules using Extended Evolutionary Algorithms. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, 1999.

[41] P. Puschner and R. Nossal. Testing the Results of Static Worst-Case Execution-Time Analysis. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS '98)*, Madrid, Spain, December 1998.

[42] A. C. Schultz, J. J. Grefenstette, and K. A. D. Jong. Adaptive Testing of Controllers for Autonomous Vehicles. In *Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology*. IEEE, 1992.

[43] A. C. Schultz, J. J. Grefenstette, and K. A. D. Jong. Learning to Break Things: Adaptive Testing of Intelligent Controllers. Naval Research Laboratory, Oxford University Press, 1995.

[44] A. Sheta. Reliability Growth Modeling for Software Fault Detection Using Particle Swarm Optimization. In *IEEE Congress on Evolutionary Computation*. IEEE, 2006.

[45] T. Shiba, T. Tsuchiya, and T. Kikuno. Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing. In *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC 04)*, IEEE, 2004.

[46] J. Stardom. Metaheuristics and the Search for Covering and Packing Arrays. Masters Thesis, Simon Fraser University, 2001.

[47] M. Tlili, S. Wappler, and H. Sthamer. Improving Evolutionary Real-Time Testing. In *Proceedings of the 2006 Conference on Genetic and Evolutionary Computation, GECCO 06*, Seattle, Washington, USA, July 8–12 2006.

[48] N. Tracey, J. Clark, and K. Mander. The Way Forward for Unifying Dynamic Test Case Generation: The Optimisation – Based Approach. In *International Workshop on Dependable Computing and Its Applications, IFIP*, 1998.

[49] N. J. Tracey, J. Clark, J. McDermid, and K.Mander. Integrating Safety Analysis with Automatic Test Data Generation for Software Safety Verification. In *Proceedings of the 17th International Conference on System Safety*, IEEE, August 1999.

[50] K. R. Walcott, M. Soffa, G. M. Kapfhammer, and R. Roos. TimeAware Test Suite Prioritization. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, ACM, New York, USA, 2006.

[51] J. Wegener, K. Grimm, M. Grochtmann, H. Sthamer, and B. Jones. Systematic Testing of Real-time Systems. In *Proceedings of the 4th European Conference on Software Testing, Analysis Review (EuroSTAR 1996)*, Amsterdam, Netherlands, December 1996.

[52] J. Wegener and M. Grochtmann. Verifying Timing Constraints of Real-Time Systems by Means of Evolutionary Testing. *Real-time Systems*, 1998.

[53] J. Wegener, R. Pitschinetz, and H. Sthamer. Automated Testing of Real-Time Tasks. In *Proceedings of the 1st International Workshop on Automated Program Analysis, Testing and Verification*, Limerick, Ireland, June 2000.

[54] J. Wegener, H. Sthamer, B. F. Jones, and D. E. Eyres. Testing Real-time Systems Using Genetic Algorithms. *Software Quality Journal*, 6(2):127–135, June 1997.

[55] Y. Zhan and J. Clark. Search-based Automatic Test-Data Generation at an Architectural Level. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2004), Lecture Notes in Computer Science (LNCS 3103)*, Springer–Verlag, Berlin, Germany, 2004.

A Degraded ILP Approach for Test Suite Reduction

Zhenyu Chen, Xiaofang Zhang and Baowen Xu
School of Computer Science and Engineering, Southeast University, Nanjing, China.
{zychen, xfzhang, bwxu}@seu.edu.cn

***Abstract*— As the cost of executing and maintaining a large test suite is always expensive, many heuristic techniques have been brought out for test suite reduction in spite of no guarantee of minimum size. The integer linear programming (ILP) approach can generate minimum test suites but it may cost exponential time. This paper proposes a degraded ILP (DILP) approach to bridge the gap between the ILP method and traditional heuristic methods. The DILP can produce a lower bound of minimum test suite and then search a small test suite close to the lower bound. An empirical evaluation of DILP is designed on Boolean specification-based testing. Four typical heuristic reduction strategies: G, GRE, H and GC are compared with DILP empirically. The experimental results show that DILP always outperforms other heuristic reduction strategies and it sometimes can guarantee the minimum size.**

I. Introduction

In a software testing process, the testing requirements firstly need to be defined from software specifications or implementations. Then, test cases are designed to satisfy the requirements manually or automatically. The test cases designed for a particular requirement may also satisfy other requirements in practice, i.e. a requirement may be satisfied by more than one test case. As a result, the constructed test suite may contain redundancy. Some subsets of the constructed test suite may still satisfy the same testing requirements. As the redundancy increases the cost of executing and maintaining the test suite, it is valuable to generate a small test suite satisfying all testing requirements.

We use $R = \{r_1, \cdots, r_m\}$ to denote the set of testing requirements which must be satisfied in the testing process. A testing requirement is said to be feasible if there is at least one test case satisfying the testing requirement. We assume that every testing requirement is feasible in this paper. A test suite is a set of test cases, denoted by $T = \{t_1, \cdots, t_n\}$. The set of all testing requirements satisfied by t is denoted by $Req(t)$. The set of all test cases satisfying r is denoted by $Test(r)$. A test suite T satisfies R if for each testing requirement r in R, there is at least one test case in T satisfying r.

T' is said to be a representative set of T if T' is a subset of T such that T' can satisfy R. A test case t is said to be 1-1 redundant to T if $Req(t) \subseteq Req(t')$ for an other $t' \in T$. $T - \{t\}$ is a representative set of T for a 1-1 redundant test case t. This is so-called 1-1 reduction strategy. A test case t is said to be essential to R if there exists $r \in R$ such that $Test(r) = \{t\}$. An essential test case t must be in every representative set.

The objective of test suite reduction is to find a small representative set for a given test suite. A minimum sized representative set is desirable. However, the problem of finding a minimum test suite is equivalent to the set covering problem, which is known to be NP-complete [8]. A test suite reduction problem can be translated into an ILP problem, then some ILP tools could be used to produce a minimum test suite [11]. However, ILP is not suitable for large test suites because it may cost exponential time. A practical approach for test suite reduction is to develop heuristic strategies in spite of no guarantee of minimum test suites. It is often referred to search based software engineering [9].

A challenge of existing heuristic methods is so-called stopping criteria. That is the testers could not estimate whether the result is good enough. Hence, they could not determine whether it needs to use expensive method (such as ILP) to improve the existing result. In this paper, a degraded ILP (DILP) method is proposed to bridge the gap between the ILP method and traditional heuristic methods. The DILP first produces a lower bound (Lb) of minimum test suites. Then it uses single-branch strategy to search a small test suite T' close to Lb efficiently. As a result, the testers can make a choice in three cases: (1) The size of T' equals to Lb then T' is a minimum one, i.e. the best result. (2) The size of T' is close to Lb then T' can also be considered as a good result. (3) The size of T' is far from Lb then it needs to use ILP or other expensive methods to improve T'.

The rest of this paper is organized as follows. In the next section, we describe some related work of test suite reduction. In section 3, we propose the DILP approach, including the preprocess by 1-1 reduction, single branch strategy and DILP algorithm. Section 4 describes an empirical evaluation on Boolean specification-based testing. Four typical heuristic strategies are compared with the DILP approach. The conclusion is drawn in the last section.

II. Related Work

The greedy strategy (denoted by G) [7] has been used in many fields of computer science including test suite reduction. M.J. Harrold et al. proposed the heuristic reduction strategy H by grouping test cases [10]. T. Y. Chen et al. proposed two

This work was supported in part by the National Natural Science Foundation of China (60425206, 60773104, 60403016, 60633010), Natural Science Foundation of Jiangsu Province (BK2005060), High Technology Research Project of Jiangsu Province (BG2005032), Excellent Talent Foundation on Teaching and Research of Southeast University, and Open Foundation of State Key Laboratory of Software Engineering in Wuhan University, Doctor subject fund of education ministry(20060286020), Jiangsu Planned Projects for Postdoctoral Research Funds (0701003B).

enhance versions of G, called GE and GRE, by combining 1-1 reduction strategy and essential strategy [2].

The above reduction strategies ignore the fact that there are some complex interrelations among testing requirements. Based on testing requirement optimization, it is possible to obtain a smaller test suite more efficiently. X. F. Zhang et al. presented a requirement optimization model to enhance the existing test suite reduction strategies G, GE, GRE and H [16]. Recently, Z. Y. Chen et al. improved it and proposed a graph contraction (denoted by GC) method for testing requirement optimization to achieve test suite reduction [6]. The experimental results shown that GC was very competitive with GRE and it always outperformed other heuristic strategies. GRE, H, ILP and genetic method also have been studied for insight into the selection of test suite reduction techniques [17].

Four typical reduction heuristics strategies: G, GRE, H and GC are compared with DILP in this paper.

- **G** is the greedy algorithm [7] for test suite reduction. It selects one test case t satisfying the maximum number of testing requirements in R and removes the satisfied requirements in $Req(t)$. And then it selects one test case satisfying the maximum number of remaining testing requirements. The selection repeats until all requirements in R are satisfied.
- **GRE** is an enhanced version of the greedy heuristic [2]. It combines the following three strategies: the essential strategy, the 1-1 reduction strategy and the greedy strategy. It is basically the alternate application of the essential strategy and the 1-1 reduction strategy. The greedy strategy is applied only when both strategies cannot be applied.
- **H** is a heuristic algorithm categorizing test cases according to different degree of 'essentialness' [10]. All requirements $r_1, \cdots, r_m$ are divided into $R_1, \cdots, R_d$. R_i denotes the set of all requirements in R that are satisfied by exactly i test cases in T. d denotes the maximum number of test cases that a requirement can be satisfied. Roughly speaking, test cases satisfy requirements in R_i are considered to be more 'essential' than those satisfy requirements in R_j for $i < j$. Clearly, H first selects test cases that satisfy requirements in R_1. And then it considers the group of unsatisfied requirements in $R_2, \cdots, R_d$ orderly and selects test cases until all requirements are satisfied.
- **GC** is a heuristic algorithm contracting testing requirements based on requirement relation graph [6]. A requirement relation graph $G(V, E)$ is constructed first based on testing requirement analysis. V is the set of testing requirements. An edge $(v, v') \in E$ if and only if there are some test cases satisfying both v and v'. Then, some graph contraction strategies are proposed to merge the vertices. As a result, a minimal set of testing requirements is obtained and test suite reduction is achieved.

In the real-world software testing, there are often multiple test criteria [1]. K. R. Walcott et al. considered the execution time of the test suite as an important cost driver [13]. S. Yoo et al. introduced the concept of Pareto efficiency to solve multi-object test suite reduction [15]. For simplicity, this paper treats test suite reduction as a single objective optimization problem.

III. Degraded ILP Approach for Test Suite Reduction

A. Preprocess of Reduction

At present, all known algorithms for NP-complete problems require time that is superpolynomial in the input size. It is unknown whether there are any faster algorithms. However, there exists some polynomial time, even linear time, algorithms which can simplify the original NP-complete problem to be a small one. Although the resulting problem is still NP-complete, it can be solved more efficiently than the original one.

The satisfiability relation between T and R could be represented as a set $S(T, R) = \{(r, t) \in R \times T : t \text{ satisfies } r\}$. Let $TS(T, R)$ denote the set of all representative sets of T w.r.t. R and $OptTS(T, R)$ denote the set of all minimum representative sets of T w.r.t. R. For a 1-1 redundant test case t in T, $OptTS(T - \{t\}, R) \subseteq OptTS(T, R)$. That is, a 1-1 redundant test case can be eliminated to simplify the satisfiability relation between R and T [4]. Similarly, a requirement r is said to be 1-1 redundant to R if there exists an other requirement r' such that $Test(r') \subseteq Test(r)$. For a 1-1 redundant requirement r, $TS(T, R - \{r\}) = TS(T, R)$. Hence, $OptTS(T, R - \{r\}) = OptTS(T, R)$ [6]. A 1-1 redundant requirements could be removed to simplify the satisfiability relation. Given a satisfiability relation $S(T, R)$, a 1-1 reduction satisfiability relation $S'(T', R')$ can be obtained by removing 1-1 redundant test cases and testing requirements one by one until there is no 1-1 redundant one in $S'(T', R')$. $S'(T', R')$ is smaller than $S(T, R)$ and $OptTS(T', R') \subseteq OptTS(T, R)$, thus the test suite reduction problem is simplified.

B. ILP Approach

In mathematics, integer linear programming (ILP) problems involve the optimization of a linear objective function subject to inequality constraints with integer variables [12]. Given a satisfiability relation $S(T, R)$, the test suite reduction problem can be translated into an ILP (actually 0-1-ILP) problem as the form [11]:

$$\text{Min } (\sum_{j}^{n} x_j) : x_j \in \{0, 1\}$$
$$\text{subject to } S \times x \geq \mathbf{1} \quad (1)$$

S is an $m \times n$ relational matrix with $s_{i,j} = 1$ if t_j satisfies r_i and $s_{i,j} = 0$ otherwise. $\mathbf{1}$ is an m-vector $(1, \cdots, 1)$. x is an n-vector $(x_1, \cdots, x_n)$ to be determined.

A naive approach of ILP is to enumerate all possible solutions, nevertheless this is feasible only for very small problems. The usual methods of ILP are implicit enumeration techniques. The "implicit" means that many solutions will

hopefully be skipped during enumeration as they are known to be non-optimal.

One of the usual implicit enumeration techniques used to solve ILP problems is Branch-and-Bound algorithm [12]. A branching strategy of 0-1-ILP is to pick a variable x_j and to replace the current problem by two subproblems, which are copies of the current problem with the variable x_j set to 0 in one and set to 1 in the other. Since the variable x_j has to take value either 0 or 1 in an optimal solution, this branching scheme guarantees that an optimal solution of the original problem will be an optimal solution of one of the two subproblems. The Bounding operation is a function that returns a bound on the optimal solution of the current subproblem. It is possible to discard some subproblems that have a bound worse than the value of the best currently known solution of the original problem.

C. Single-Branch Strategy

The Branch-and-Bound algorithm used to solve ILP problems may create exponential subproblems in the worst case, because it creates two subproblems for each variable. The basic idea of degraded ILP (DILP) is the single-branch strategy, i.e. to select only one most possible subproblem for each variable. Hence, DILP creates at most n subproblems for n variables.

The LP relaxation of an ILP problem is obtained by removing the integrality constraints on the variables. LP problems can be solved in the polynomial time, whereas ILP problems are NP-hard in general [12]. Since the feasible solutions of the ILP problem are all feasible for the LP problem, LP solution also provides a lower bound on the optimal value of the ILP problem. If the solution of the relaxation has integer components, then it also solves the ILP problem fortunately.

Given a satisfiability relation matrix S, an LP relaxation of the ILP problem is formed as $Min(\sum_j^n x_j)$ subject to $S \times x \geq \mathbf{1}$ with $x \in [0, 1]$. The LP relaxation is easy to solve by some algorithms, such as simplex algorithm [12]. A feasible solution v can be output by some LP algorithms, in which v_j is the value of x_j. If v_j is not an integer for some j, then v is not a solution of the original ILP problem. However, the value v_j could be considered as the possibility of optimal solution of ILP problem. For example, $v_i = 0.9$ and $x_j = 0.3$ indicate x_i will more potentially be 1 than x_j in the optimal solutions of ILP problem.

A single-branch strategy of 0-1-ILP is to pick a variable x_j with v_j close to 1 and to replace the current problem by a restricted problem, which is a copy of the current problem with fixing the variable x_j to 1. A challenge of the single-branch strategy is which variable should be fixed firstly. A natural choice is to fix the variable x_j with a high value v_j to 1. Note that the variable x_j will contribute each testing requirement r_i only if $s_{i,j} = 1$, i.e. t_j satisfies r_i. Hence a more suitable metric is introduced as follows.

$$d_j := \sum_i^m s_{i,j} * v_j \tag{2}$$

D. DILP Algorithm

The pseudo-code of degraded ILP (DILP) approach is shown in algorithm 1.

	Algorithm 1: **DILP**(S)
1:	1-1 reduction S;
2:	$v = LP(S)$;
3:	$Lb = Int(\sum_j^n v_j)$;
4:	while 1
5:	if each v_j is an integer
6:	return v and Lb;
7:	end if
8:	Maxd=0;
9:	for each j
10:	if $v_j == 1$
11:	Fix x_j to 1;
12:	else
13:	Compute d_j;
14:	if $d_j >$Maxd
15:	Maxd=d_j;
16:	$k = j$;
17:	end if
18:	end if
19:	end for
20:	Fix x_k to 1;
21:	$v = LP(S)$;
22:	end while

Firstly a satisfiability relation matrix S is input in the procedure DILP. We use 1-1 reduction as a preprocess of DILP until there is no 1-1 redundant test case and no 1-1 redundant testing requirements (line 1). An LP relaxation of ILP problem in equation (1) will be solved by LP algorithms (line 2). The sum of result v can be considered as a lower bound Lb of minimum test suite (line 3), because any ILP solution is also an LP solution. If each v_j is an integer, then v is an optimal solution of the original ILP problem (line 5-7). Otherwise, each x_j will be fixed to 1 for $v_j = 1$ (line 11). We calculate the metric d_j for each remained x_j, i.e. $v_j < 1$, (line 13). The maximal d_k is selected (line 15-16) and x_k is fixed to 1 (line 20). It is formed as a restricted LP problem and it will be solved by LP algorithms again (line 21). The statements in loop (line 4-22) are repeated until each v_j is an integer and v and Lb are output. The loop will stop in at most n times, because at least one x_j is fixed in each iteration and there are total n variables in x.

IV. Empirical Evaluation

In this section, an experiment on a suite of Boolean specifications from TCAS II is designed and implemented to evaluate the DILP approach. Four heuristic reduction strategies: G, GRE, H and GC, are also compared with DILP.

A. Experiment Design

Given a Boolean specification P, an implementation expression may be a mutant M by making simple syntactic changes to P. A test case t is an assignment for all variables. $P(t)$

TABLE I

EXPERIMENTAL SUBJECTS

No.	$\|V\|$	$\|L\|$	$\|T\|$	$\|LNF\|$	$\|LRF\|$
1.	7	23	62	21	235
2.	9	36	113	36	539
3.	12	46	2970	46	981
4.	5	5	29	5	40
5.	9	20	392	20	302
6.	11	28	142	25	468
7.	10	21	210	21	362
8.	8	17	36	17	228
9.	7	10	16	10	120
10.	13	15	256	15	353
11.	13	20	2188	20	443
12.	14	17	4290	17	438
13.	12	13	1731	13	279
14.	7	12	107	12	142
15.	9	18	372	18	266
16.	12	37	2834	37	794
17.	11	11	1033	11	220
18.	10	11	584	11	198
19.	8	9	116	9	126
20.	7	8	24	8	96

TABLE II

EXPERIMENTAL RESULTS OF LNF

No.	$\|T\|$	$Rst_i(*)$					Bst_i
		G	GRE	H	GC	DILP	ILP
1	62	7	7	8	7	7	7
2	113	6	6	6	6	6	6
3	2970	10	10	10	10	10	10
4	29	2	2	2	2	2	2
5	392	6	5	6	5	5	5
6	142	4	4	5	4	4	4
7	210	3	3	4	4	3	3
8	36	2	2	2	2	2	2
9	16	2	2	2	2	2	2
10	256	3	3	4	3	3	3
11	2188	5	5	5	5	5	5
12	4290	4	4	5	4	4	4
13	1731	3	3	3	3	3	3
14	107	4	4	4	4	4	4
15	372	4	4	4	4	4	4
16	2834	7	7	9	7	7	7
17	1033	3	3	3	3	3	3
18	584	3	3	3	4	3	3
19	116	2	2	3	2	2	2
20	24	2	2	2	2	2	2

and $M(t)$ denote the values of P and M evaluated by the test case t, respectively. In general, a mutant M may happen to be logically equivalent to the specification P and hence it cannot be distinguished by any test case. A non-equivalent mutant is called a fault and an equivalent mutant is not regarded as a fault. A fault M is said to be killed (or detected) by a test case t if $M(t) \neq P(t)$. That is, t is a satisfying assignment of $M \oplus P$ ($\oplus$ is the exclusive-or operator). For each mutant M_i, a testing requirement is formed as a Boolean expression $r_i = P \oplus M_i$. A testing requirement r_i is feasible if and only if r_i is satisfiable. It is not difficult to see that the number of test cases is finite. Before test suite reduction, all test cases, i.e. satisfying assignments, could be generated to construct an initial test suite.

Our experimental analysis was done using software that was specifically designed and implemented for the purpose above. The software allows the analysis of a given Boolean specification. The experimental steps involved in the empirical analysis were as follows:

1. Select the subject Boolean specifications.
2. Generate the mutants and testing requirements.
3. Construct the initial test suites.
4. Reduce the test suites using the reduction strategies.

We used the set of 20 Boolean specifications, which were originated from the specification of an aircraft collision avoidance system called TCAS II [14]. Two of fault classes, LNF and LRF [5], were considered in the experiment. Literal Negation Fault (LNF): A literal is replaced by its negation, e.g., $(a \wedge b) \vee (\neg b \wedge c)$ implemented as $(a \wedge b) \vee (b \wedge c)$ with $\neg b$ replaced by b. Literal Reference Fault (LRF): A literal is replaced by another literal that appears in the decision, e.g., $(a \wedge b) \vee (\neg b \wedge c)$ is implemented as $(a \wedge b) \vee (\neg a \wedge c)$ with $\neg b$ replaced by $\neg a$.

Each mutant M of P was generated first, and then the testing requirement was formed as $P \oplus M$. $|LNF|$ and $|LRF|$ denote the numbers of feasible testing requirements and the infeasible testing requirements are ignored. $|V|$ and $|L|$ denote the number of variables and the number of literal occurrences in Boolean specifications, respectively. The number of all test cases is $2^{|V|}$. $|T|$ denotes the number of test cases used in LNF and LRF. The details were shown in Table I.

The number of LNF mutants equals to L_i and the number of LRF mutants equals to $L_i \cdot (|V_i| - 1) \cdot 2$. However, the number of feasible testing requirements may be less than the number of mutants, because there may be some equivalent mutants for P. For example, $|R_1| = 21 < 23 = |L_1|$ for LNF in Table I. The size of test suite is $2^{|V_i|}$ in the worst case. However, the real size was always much smaller than $2^{|V_i|}$. For example, $|T_8| = 36 < 2^8 = 256$ for LNF in Table I.

B. Experimental Results and Analysis

The effectiveness of each reduction strategy was measured by computing the size of resulting test suites for each Boolean specification. To a further comparison, four typical reduction strategies: G, GRE, H and GC, were also implemented for test suite reduction. $Rst_i(*)$ denotes the size of resulting test suite for the no. i Boolean specification using the reduction strategy $*$. A minimum test suite is desirable for test suite reduction. We computed the size of minimum test suite for each Boolean specification using ILP method [11], denoted by Bst_i. $|T|$ denotes the size of initial test suite for LNF or LRF. The detail experimental results of LNF and LRF were shown in Table II and III, respectively. Main observations of the empirical analysis were made as follows.

Effectiveness of DILP Approach.

The 1-1 reduction was implemented first as a preprocess of reduction until there is no 1-1 redundant testing requirements and no 1-1 redundant test cases. The experimental results of evaluation for 1-1 reduction were shown in Fig. 1 and 2. The number of original requirements and test cases were

TABLE III

EXPERIMENTAL RESULTS OF LRF

No.	$\|T\|$	$Rst_i(*)$					Bst_i
		G	GRE	H	GC	DILP	ILP
1	62	19	18	19	18	18	18
2	113	26	26	26	26	26	26
3	2970	36	36	38	35	33	31
4	29	5	5	5	5	5	5
5	392	18	15	16	15	15	15
6	142	26	25	27	25	25	25
7	210	21	21	20	19	19	19
8	36	18	18	18	18	18	18
9	16	12	12	12	12	12	12
10	256	16	16	17	16	15	15
11	2188	19	20	23	21	18	17
12	4290	16	17	18	17	15	15
13	1731	13	13	15	13	13	13
14	107	14	13	14	13	12	12
15	372	17	17	17	16	16	16
16	2834	28	28	36	28	25	25
17	1033	13	13	12	11	11	10
18	584	13	12	14	13	11	10
19	116	12	11	11	10	9	8
20	24	10	10	10	10	10	10

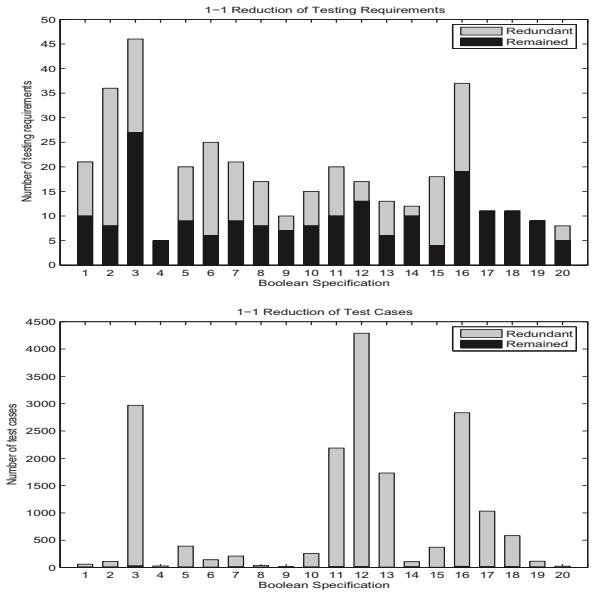


Fig. 1. Evaluation of 1-1 Reduction for LNF

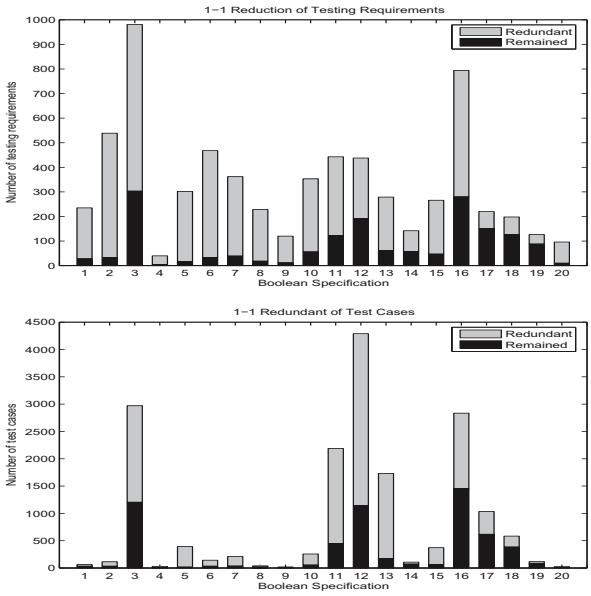


Fig. 2. Evaluation of 1-1 Reduction for LRF

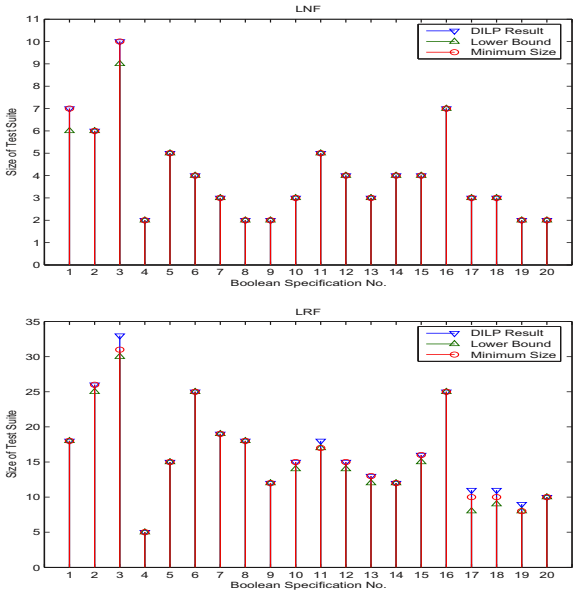


Fig. 3. Evaluation of DILP Approach

represented as the whole plots. The numbers of 1-1 redundant requirements and test cases were represented as the gray plots. The number of remained requirements and test cases were represented as the black plots. As is evident from Fig. 1 and 2, the results of 1-1 reduction were very inspiring, particularly for 1-1 reduction of test cases for LNF. One reason may be testing requirements are much less than test cases for LNF, as a result, there are many 1-1 redundant test cases.

The lower bound was output from LP relaxation by LP algorithms. Then DILP would search a feasible solution close to the lower bound. Note that the lower bound may not be the greatest lower bound, i.e. it might not be reached realistically. The greatest lower bound, i.e. minimum size, of reduced test suite can be calculated by ILP algorithms. However, if the result of DILP equals to the lower bound, then the DILP result must be a minimum sized test suite. That is, DILP approach solves the ILP problem fortunately. The comparison of DILP result, lower bound and minimum size was shown in Fig. 3. The experimental results of DILP approach were very inspiring. In the case of LNF, DILP could generate minimum test suite for all Boolean specifications. The lower bounds of no. 1 and 3 Boolean specifications were not the greatest lower bounds, hence we could not conclude that the DILP results were minimum ones despite they were so actually. In the case of LRF, DILP could generate minimum ones for 15 Boolean

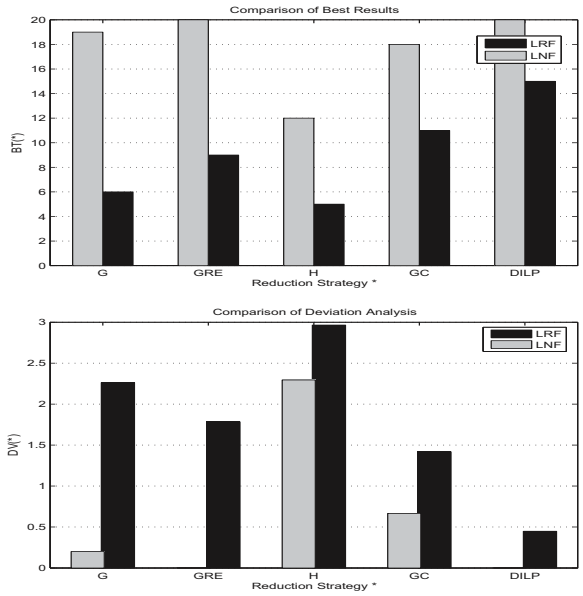


Fig. 4. Comparison of Heuristic Reduction Strategies

specifications. DILP can guarantee to generate the minimum ones for 10 Boolean specifications, because they reached the lower bounds.

Comparison of Heuristic Reduction Strategies.

The detail experimental results of different reduction strategies were shown in Table II and III. To facilitate the comprehension for readers, two evaluation metrics were introduced to show the results of comparison. $BT(*)$ denotes the times of reduction strategy $*$ generating a minimum test suite. That is,

$$BT(*) = \sum_{i=1}^{20} (Rst_i(*) == Bst_i) \tag{3}$$

To a further comparison, standard deviation analysis was introduced to quantify the goal of test suite reduction. The formalization is described as follows.

$$DV(*) = \sum_{i=1}^{20} \frac{Rst_i(*) - Bst_i}{Bst_i} \tag{4}$$

For a reduction strategy, a higher value of $BT(*)$ suggests that it obtains better results with respect to the other reduction strategies. A lower value of $DV(*)$ suggests that it obtains better results with respect to the other reduction strategies. The results of four typical heuristic reduction strategies and DILP approach were shown in Fig. 4. For $BT(*)$, the DILP approach wined the best score among all reduction strategies, for both LNF and LRF. The results of $DV(*)$ conformed to the ones of $BT(*)$. In general, the results of LRF was more significant than the results of LNF, because the numbers of requirements and test cases of LRF are much more than the ones of LNF. The experimental results shown that DILP was always outperformed other heuristic reduction strategies.

V. Conclusion and Future Work

In this paper, a degraded ILP (DILP) approach was proposed to bridge the gap between the ILP method and traditional heuristic methods. DILP could produce a lower bound of minimum size and then search a feasible solution close to the lower bound. The experimental results shown that DILP always outperformed typical heuristic reduction methods and it can sometimes guarantee the generation of minimum test suite. However, the complexity of LP algorithms was higher than the typical heuristic methods, although LP problems can be solved in the polynomial time. The comparison of time cost need to be discussed further. The empirical evaluation is still very primary, it could not draw rich conclusions for DILP and other heuristic reduction strategies. In the future, large scale testing objects [17] and simulation data [3] would be studied for insight into the selection of test suite reduction techniques.

References

[1] J. Black, E. Melachrinoudis, and D. Kaeli. Bi-criteria models for all-uses test suite reduction. In *Proceedings of 26th International Conference on Software Engineering*, pages 106–115. ACM Press, 2004.

[2] T. Y. Chen and M. F. Lau. A new heuristic for test suite reduction. *Information and Software Technology*, 40(5):347–354, 1998.

[3] T. Y. Chen and M. F. Lau. A simulation study on some heuristics for test suite reduction. *Information and Software Technology*, 40(13):777–787, 1998.

[4] T. Y. Chen and M. F. Lau. On the completeness of a test suite reduction strategy. *The Computer Journal*, 42(5):430–440, 1999.

[5] Z. Y. Chen, B. W. Xu, and C. H. Nie. Comparing fault-based testing strategies of general Boolean specifications. In *Proceedings of the 31st International Computer Software and Applications Conference*, pages 621–622. IEEE Computer Society Press, 2007.

[6] Z. Y. Chen, B. W. Xu, X. F. Zhang, and C. H. Nie. A novel approach for test suite reduction based on requirement relation contraction. In *Proceedings of the 23rd Annual ACM Symposium on Applied Computing*, pages 390–394. ACM Press, 2008.

[7] T. H. Connen, R. L. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theorey of NP-completeness*. W. H. Freeman and Company, 1979.

[9] M. Harman. The current state and future of search based software engineering. In *Proceedings of Workshop on Future of Software Engineering (FOSE'07)*, pages 20–26. ACM/IEEE, 2007.

[10] M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology*, 2(3):270–285, 1993.

[11] J. G. Lee and C. G. Chung. An optimal representative set selection method. *Information and Software Technology*, 42(1):17–25, 2000.

[12] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[13] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos. Time aware test suite prioritization. In *Proceedings of International Symposium on Software Testing and Analysis (ISSTA'06)*, pages 1–12. ACM Press, 2006.

[14] E. Weyuker, T. Goradia, and A. Singh. Automatically generating test data from a Boolean specification. *IEEE Transactions on Software Engineering*, 20(5):353–363, 1994.

[15] S. Yoo and M. Harman. Pareto efficient multi-objective test case selection. In *Proceedings of International Symposium on Software Testing and Analysis (ISSTA'07)*, pages 140–150. ACM Press, 2007.

[16] X. F. Zhang, B. W. Xu, C. H. Nie, and L. Shi. An approach for optimizing test suite based on testing requirement reduction. *Journal of Software*, 18(4):821–831, 2007.

[17] H. Zhong, L. Zhang, and H. Mei. An experimental comparison of four test suite reduction techniques. In *Proceedings of 28th International Conference on Software Engineering*, pages 636–640. ACM Press, 2006.

A Meta-Model to Support Regression Testing of Web Applications

Yanelis Hernandez, Tariq M. King, Jairo Pava and Peter J. Clarke
School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
email: {yhern004, tking003, jpava001, clarkep}@cis.fiu.edu

Abstract

As businesses strive to keep pace with the rapid evolution of web technologies, their efforts to maintain automated regression testing strategies are being hindered. Technological migration of a web application can lead to test scripts becoming incapable of validating the migrated application due to differences in the testing platform. Regression tests that are still applicable to the application would therefore have to be re-written to be compatible with the new technologies. In this paper, we apply a model-driven approach to the development of automated testing scripts for validating web applications from the client-side. We define a meta-model using UML 2.0 profiles, and describe the model transformations needed to automatically port regression tests to various platforms. A prototype of the test implementation for an e-commerce application is presented to demonstrate the feasibility of the approach.

Keywords: Regression Testing, MDSD, UML Profiles, Web Application

1 Introduction

The many advances in web technologies has led to the development of web applications that compete in solution areas that traditional software previously addressed. Web applications are no longer simple streams of static web pages, but instead provide a collection of interactive services with the added flexibility, mobility, and connectivity of the Internet. These characteristics have made web-based solutions highly attractive to businesses, and this has led to the creation of many development and testing tools to support web programming [5, 16, 17, 18]. However, a negative consequence of these advancements is the persistent growth in the complexity of web applications, and the rapid evolution of their supporting technologies.

Software testing is a very costly and time-consuming endeavor. Some studies indicate that the cost of software testing may account for between fifty to seventy-five percent of total development costs [7, 8]. In addition, testing costs tend to exceed those of design and implementation, and therefore the methodologies and tools employed at these stages are pertinent to the development of affordable quality software.

Automation is an effective way to reduce time and costs of software testing, and so many businesses conduct their testing process with some degree of automation. The level of automation of software testing typically exists at the test script level. Software testers encode a set of test cases for the application in a scripting language, and use the script as input to an automated testing tool which executes the tests. If subsequent changes are made to the system, the test script provides a means for automatically performing regression testing to determine whether new errors were introduced into previously tested components [6].

Script-level test automation becomes problematic when an application migrates to include technologies that are not supported by the testing tool currently being utilized. Regression tests that are still applicable to the migrated application therefore have to be re-written in the scripting language of a new testing tool, thereby defeating the purpose of test automation.

The model-driven software development (MDSD) paradigm emphasizes the use of models and model transformations to generate executable code for a specific platform. In this paper, we apply MDSD to the generation of an automated testing script for validating the client-side of a web application. To address the aforementioned problem of script-level automation, we propose that the test set for the web application be designed as a platform independent model which can be automatically transformed into a platform specific automated testing script.

The main contributions of this work are that it: (1)

presents a model-driven approach to the design and development of automated testing scripts to validate a web application; (2) provides meta-models for a subset of web-based development and testing technologies using UML 2.0 [12] profiles; and (3) elaborates on a case study developed using the proposed modeling approach to support testing.

This paper is organized as follows: the next section contains background information on web-based technologies, regression testing, and meta-modeling. Section 3 presents the proposed approach to support testing a migrated web application. Section 4 contains the meta-models used in our approach and describes the generation of the testing script. Section 5 provides the details of the case study. Section 6 presents related work, and in Section 7 we give concluding remarks and discuss future work.

2 Background

In this section we provide background information on the technologies commonly used to develop web applications. We then discuss the technique of regression testing, including tool support for automatically validating web applications. Approaches to meta-modeling are also described in this section.

2.1 Web-Based Technologies

There are two broad categories of web programming technologies used to develop web applications – *client-side* and *server-side* [4]. Client-side scripting technologies involve the use of a web-browser on the client machine to perform operations without having to communicate with the server. This type of scripting is generally used to dynamically modify the behaviors within a specific web page in response to user input [4]. Popular examples of client-side scripting technologies include [16, 18]: HTML and Javascript.

In contrast, server-side technologies perform operations on the web server instead of on the client machine. They are preferable when operations utilize information that is not available on the client, or when data storage from the client to the server is needed [4]. Dynamic operations on the server-side may involve changing the web page supplied to the client, or providing a new sequence of web pages to the browser. Active Server Pages (ASP) [4] and Hypertext Preprocessor (PHP) [17] are two examples of server-side scripting languages commonly used to develop web applications.

Many web technologies can be integrated with others and hence web applications usually employ a myriad of technologies on both the client and the server. In essence, the classification of a scripting language depends on its implementation within the web application. For example, Flash [1] technologies may be implemented on the client using companion technologies such as HTML [18], or on the server by providing synchronized updates to the client.

2.2 Regression Testing

Software testing is the process of operating a software system under specified conditions, recording the results, and making an evaluation of some aspect of the software [10]. Testing is particularly useful for validating changes made to a system during software maintenance or evolution. Regression testing refers to rerunning test cases to determine whether or not new errors have been introduced into previously tested code [3]. In an effort to reduce costs, many testing strategies employ automated tools to support the performance of regression tests on modified software systems.

There has been a rapid growth of tools to support testing web applications on both the client and server sides. HTMLUnit [5] simulates the behavior of a web browser by providing an API to interact with web pages. Functional testing tools such as TestSmith provide facilities for simulating mouse and keyboard events and hence can be used on the client-side to test Flash applications. On the server-side, PHPUnit [13], a member of the xUnit family of testing frameworks, is a unit testing solution for PHP [17].

2.3 Meta-Modeling

Model-driven software development (MDSD) focuses on the combined use of software models and associated transformations to build complete software systems. This typically involves the use of a source model or Platform Independent Model (PIM), and a target model or Platform Specific Model (PSM) [15]. The PIM does not rely on any specific technological platform that could be used to implement the software, and therefore represents the essence of the solution. A model transformation language can then be used to transform the PIM into a PSM that is executable on the target platform [15].

A technique known as *meta-modeling* is used to ensure the consistency of models during transformation. This involves defining the abstract syntax of models and the interrelationships between the model elements [14]. Meta-modeling should consist of orthogonal dimensions that support two forms of instantiation [2]: *linguistic* – relates to the language definition, and *ontological* – relates to the domain definition. In this paper

the ontological meta-modeling will be implemented using UML 2.0 [12] profiles.

3 MDSD Approach to Support Testing

In this section we define the scope of the problem being addressed in this paper. We then present our approach which applies MDSD to the generation of an automated testing script for validating a web application.

3.1 Problem Definition

Automating the process of testing a web application involves developing a script that can be recognized by a testing tool, which then applies predefined test cases to the application under test. This is depicted in the left-hand portion of Figure 1, where a test script `TS1` validates an application under test `AUT1`; both of which can be thought of as targeting the same set of web technologies `WT1`.

The migration from `AUT1` to `AUT2` in Figure 1 represents when a business updates their application to include a new set of web technologies, labeled `WT2`. However, this migration usually leads to the test script `TS1` becoming incapable of validating the application `AUT2`. Therefore, regression tests that are still applicable to `AUT2` would have to be re-written in a new test script `TS2` to be compatible with the testing tool for `WT2`.

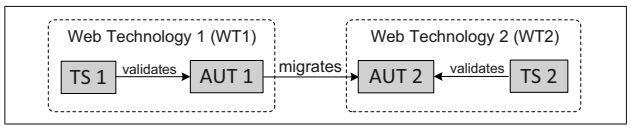


Figure 1. Testing a migrated web application.

3.2 Model-Driven Solution

Our approach harnesses the power of MDSD to automatically generate the testing script for a web application that has migrated to new technologies. Figure 2 shows the models and transformation processes used in our approach. A key aspect of our methodology is the use of a test script generator, shown at the center of Figure 2, to produce platform specific tests for the migrant web applications. Inputs to the generator are represented by dotted lines, while solid lines are used to represent the output.

First, a platform independent model, representing the essence of the test set for the web application, is input to the generator. This PIM is then combined with a model of the constructs used for testing a particular set of web technologies. For example, in Figure 2, `PI Test Set` would be combined with `WT Model 1`, which models constructs for testing HTML and Javascript, to automatically generate an HTMLUnit script `PS Test Set 1`. If the application later migrates to a new set of web technologies (e.g., Flash and Actionscript), the test set can also migrate automatically by combining `PI Test Set` and `WT Model 2`.

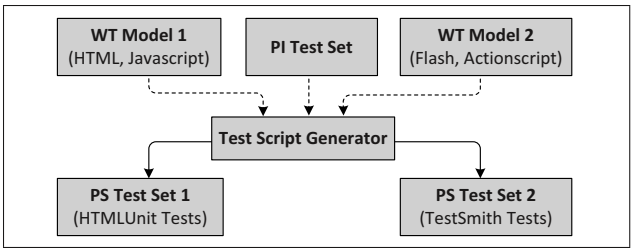


Figure 2. Model-driven test script generation.

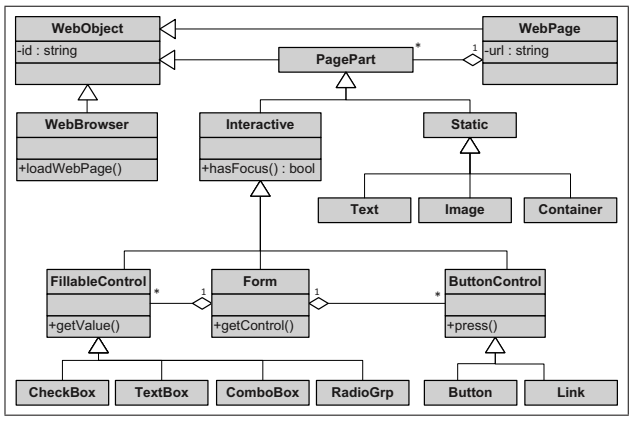


Figure 3. Conceptual model of a web interface.

4 Meta-Models

In this section we provide a conceptual model depicting abstractions for a subset of the visual elements of a web application. We then present a UML 2.0 profile of a meta-model to support testing web applications based on the conceptual model.

4.1 Conceptual Model of a Web Interface

Figure 3 shows a conceptual model for the user interface of a web application. The purpose of the model is to provide abstractions for the visual elements of the web application that are relevant to testing. At the top of the hierarchy of object types is the `WebObject` (top-left of Figure 3), which is a general representation for any element of the web interface. These object types include web pages and the elements contained within them, which are classified as follows: (1) *interactive* – allows for dynamic user interaction, e.g., forms,

Stereotype	Base Class	Tagged Value	Constraints
<<TestSet>>	Class	id: String	May only declare instances of classes stereotyped TestCase. id is unique.
<<TestCase>>	Class	id: String	May only declare instances of classes stereotyped TestSection, and be associated with at most one instance each of classes stereotyped Setup, Precondition, Input, Postcondition, and Rollback. id is unique.
<<TestSection>>	Class	id: String	id is unique.
<<TestCommand>>	Class	id: String	May be associated with at most one instance of a class stereotyped TestSubject. id is unique.
<<WebObject>>	Class	id: String	id is unique.
<<TestSubject>>	WebObject		
<<Setup>>	TestSection		May only declare instances of classes stereotyped CreateCommand.
<<Precondition>>	TestSection		May only declare instances of classes stereotyped InputCommand.
<<Input>>	TestSection		May only declare instances of classes stereotyped InputCommand.
<<Postcondition>>	TestSection		May only declare instances of classes stereotyped AssertCommand.
<<Rollback>>	TestSection		May only declare instances of classes stereotyped DestroyCommand.
<<CreateCommand>>	TestCommand		
<<InputCommand>>	TestCommand		
<<AssertCommand>>	TestCommand		
<<DestroyCommand>>	TestCommand		
<<contains>>	Association		Connects instances of TestSet with instances of TestCase, and instances of TestCase with instances of TestSection
<<manipulates>>	Association		Connects instances of Command with instances of TestSubject.

Table 1. UML profile of the test model for a web application.

textboxes, buttons; and (2) *static* – remains fixed regardless of external stimuli, e.g., text, images, tables.

An interesting aspect of the model is that the type WebBrowser is also derived from WebObject. This is because the browser allows users to break the normal flow of control of the application, and testing should address such scenarios. For example, a user may press the Back button of the browser during the execution of the application causing unexpected results [19]. In addition, the WebBrowser type facilitates changes to the browser configuration, and allows testing to simulate the use of a specific browser; both of which can affect the behavior of the web application.

4.2 Meta-Model to Support Testing

The UML 2.0 profile for the test model of a web application is shown in Table 1. It consists of four kinds of artifacts indicated by the column headings (from left to right): (1) stereotype – represents specific meta-classes; (2) base class – denotes an extension relationship from a UML meta-class or inheritance from another stereotype; (3) tagged value – defines attributes of the stereotype; and (4) constraints – enforce restrictions on how the meta-model may be used. For example, in Row 1 of Table 1 the stereotype TestSet extends of the UML meta-class named Class; contains the tagged value id of type String; and may only contain variables of a class whose stereotype is TestCase. The constraints for this table entry also specify that the tagged value id should only hold unique values. It should be noted that the stereotype TestSubject in Row 5 extends WebObject, which is the base class from the conceptual model of a web interface presented in Subsection 4.1.

5 Case Study

In this section we present a case study developed as an initial proof of concept realization of our methodology. We first outline the features of the application, and describe the technologies and test support tools required to setup the experiment. We then provide details on a test set implementation that uses the proposed approach, including the generation of test scripts. The findings and limitations of the study are also discussed in this section.

5.1 FastBooks Application

FastBooks is a small e-commerce application for purchasing college textbooks on-line. Users may choose to purchase their textbooks in three different formats (Print, Audio, or Electronic), and then submit their billing and shipping information for validation. Two versions of the FastBooks application were developed to set up the scenario of a business migrating from one web platform to another that uses technologies unsupported by the current testing tool.

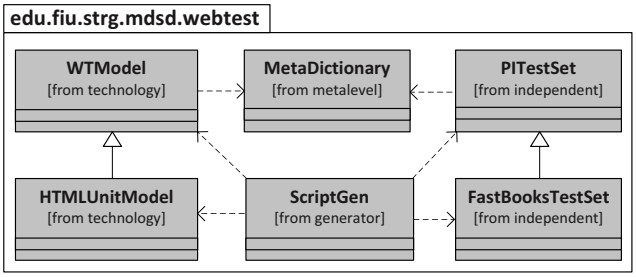


Figure 4. Minimal class diagram of the prototype.

The first version of the application was developed using HTML 4.01 and Javascript on the client-side, and implemented its automated testing using HTMLUnit 1.4. The second version of the application was developed using Flash 9 and Actionscript 3.0 on the client-side. Both versions used PHP 5.25 on the server-side for transferring data to and from persistent storage, while Apache HTTP Server 2.2 provided the web server functionality.

5.2 Test Implementation

We developed a prototype in Java 5.0 to implement the model-driven testing solution presented in SubSection 3.2. First, test cases for the FastBooks application were designed using boundary value analysis and equivalence partitioning techniques. The initial test set, consisting of 12 test cases, was then encoded using the constructs and rules of the testing meta-model proposed in this paper.

The package labeled `edu.fiu.strg.mdsd.webtest` in Figure 4 shows the main communicating classes from various sub-packages of the prototype. The types from the conceptual model of a web interface were stored in the class `MetaDictionary`, which was used to design generalized classes for modeling the web testing technology `WTModel` and platform independent test set `PITestSet`. These two classes were then specialized to create objects for holding the HTML and Javascript testing constructs, as well as the test cases designed for the FastBooks application; represented by the classes `HTMLUnitModel` and `FastBooksTestSet` respectively.

5.3 Generation of Test Scripts

The class `ScriptGen` in Figure 4 is responsible for iterating through `FastBooksTestSet`, and generating a test script using the platform specific constructs in `HTMLUnitModel`. This is achieved by retrieving the test case definitions that are represented as abstract test commands, along with variable names and their associated values. These abstract test commands are then mapped to the HTMLUnit constructs that also contain placeholders for the variable names and values. The generator then overwrites the placeholders with the actual variable names and values stored in `FastBooksTestSet`, and appends the completed instruction to the output file `FastBooks.htmlUnit`.

5.4 Discussion

The purpose of the case study was to demonstrate the feasibility of applying a model-driven approach to the design and development of testing scripts for a web application. Implementing the prototype gives credence to the claim that the strategy can be used to define platform independent tests, and convert them into scripts for an automated testing tool. All of the base test cases developed for the case study were successfully transformed into syntactically correct HTMLUnit tests. This suggests that the abstract constructs used in the current prototype were therefore sufficient for representing the platform specific constructs required to validate the FastBooks application. Although the current version of the prototype does not implement the technology model for Flash, this could be easily incorporated by extending the generalized classes provided by the infrastructure.

Conducting the study also provided us with insight into the intricacies of developing a framework for the proposed approach. Although the FastBooks test set only required the prototype to address a limited number of test scenarios, designing the framework to maintain independence among the test model, technology model, and script generator was very challenging. Limitations of the current prototype include coverage of only a subset of web controls and widgets, and manual detection of model constraint violations. However, the latter could be solved through the use of model-driven architecture tools such as the Eclipse Modeling Framework.

6 Related Work

There has been great effort in the research community to assure the quality of web applications through effective testing methodologies. However, most of the work that focuses on the use of models for testing web applications relies heavily on specific platforms for the creation of their models. In contrast, our approach uses a platform independent test model to facilitate the generation of test scripts.

The work presented by Li et al. [11] is most closely related to our work. It proposes a model-driven testing methodology for web applications. A model of the

web application is built to describe the system under test, and test cases are developed based on that model. Test scripts are generated from the test cases and executed by a test engine. Our approach differs from [11] in that we consider the negative impact that migration has on the ability to automatically validate web applications. Therefore, the technique proposed in this paper for modeling the technological constructs could be used in [11] to provide a more extensible solution.

The Object Management Group (OMG) [12] extended UML with testing concepts such as test architecture, test data, and test behavior. Similar to the profile presented in this paper, the UML Testing Profile 1.0 is based on the UML 2.0 specification and provides a modeling language that can be used to design, visualize, and specify the artifacts of a test system. The meta-models presented by [12] encapsulate a broad view of testing as a process. However, in this paper we focus on defining a profile for the structure of a test set for the client-side of a web application, and leverage the resulting models to generate test scripts.

Heckel et al. [9] present an approach for testing web applications designed with a model-driven approach. Design patterns such as Bridge and Proxy are used to execute the same test cases in a local and distributed testing environment, respectively. Their methodology takes advantage of the separation of PIMs and PSMs on both the model-level and implementation-level. Although the scope of our work does not include test execution, the strategies in [9] could be used to design a test harness for scripts generated by our approach.

7 Concluding Remarks

In this paper we presented a model-driven technique for designing platform independent tests for validating web applications. Our approach leverages these test case models for the generation of automated test scripts, thereby addressing the problems associated with technological migration of the web application. An e-commerce application was used as the basis of our study and a prototype was implemented.

Future work calls for deeper investigation into the problem by: (1) extending the prototype to include the technology model for Flash and Actionscript; (2) formulating additional test scenarios for the application used in the case study, and (3) developing a platform independent model for tests that target server-side scripting languages.

8. Acknowledgements

The authors would like to thank Dr. Robert France for his contribution to this work.

References

[1] Adobe Systems Inc. Flash 9, April 2007. `http://www.adobe.com/products/flash/` (Mar. 2008).

[2] C. Atkinson and T. Kühne. Model-driven development: A metamodeling foundation. *IEEE Softw.*, 20(5):36–41, 2003.

[3] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, New York, second edition, 1990.

[4] D. Buser, C. Ullman, J. Duckett, J. Kauffman, J. T. Llibre, and D. Sussman. *Beginning Active Server Pages 3.0*. Wrox Press Ltd., Birmingham, UK, UK, 2000.

[5] Gargoyle Software Inc. HTMLUnit 1.14, Jan 2008. `http://htmlunit.sourceforge.net/` (Mar. 2008).

[6] T. L. Graves, M. J. Harrold, J.-M. Kim, A. Porter, and G. Rothermel. An empirical study of regression test selection techniques. *ACM Trans. Softw. Eng. Methodol.*, 10(2):184–208, 2001.

[7] B. Hailpern and P. Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12, 2002.

[8] M. J. Harrold. Testing: a roadmap. In *ICSE - Future of SE Track*, pages 61–72, 2000.

[9] R. Heckel and M. Lohmann. Towards model-driven testing. *Electr. Notes Theor. Comput. Sci.*, 82(6), 2003.

[10] IEEE Computer Society. Std 610.12-1990(r2002): Standard glossary of software engineering terminology. Technical report, 2002.

[11] N. Li, Q. qin Ma, J. Wu, M. zhong Jin, and C. Liu. A framework of model-driven web application testing. In *COMPSAC (2)*, pages 157–162, 2006.

[12] Object Management Group. Unified modeling language. `http://www.uml.org/` (Mar. 2008).

[13] Sebastian Bergmann. PHPUnit 3.2.15, Feb. 2008. `http://www.phpunit.de/` (Mar. 2008).

[14] S. Sendall and W. Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Softw.*, 20(5):42–45, 2003.

[15] T. Stahl, M. Voelter, and K. Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[16] Sun Microsystems, Inc. JavaScript, December 1995. `http://java.sun.com/javascript/` (Mar. 2008).

[17] The PHP Group. PHP 5, Nov. 2007. `http://www.php.net/` (Mar. 2008).

[18] W3C. HyperText Markup Language 4, December 1999. `http://www.w3.org/TR/REC-html40/` (Mar. 2008).

[19] Y. Wu and J. Offutt. Modeling and testing web applications. Technical report, GMU ISE Technical ISE-TR-02-08, November 2002.

Service Granularity Effects in SOA

Ned Chapin
InfoSci Inc., Box 7117
Menlo Park CA 94026-7117
NedChapin@acm.org

Abstract

In service-oriented architecture (SOA), what are chosen to be the services is a matter of local choice. That choice includes what degrees of service granularity are used. The degrees range from fine grained to very large grained, and usually are not uniformly used. The effects of the choices of service granularity are major. The choices effect the fabric and outcome of the SOA design and implementation, as experienced by managers, software users, and the information systems personnel developing and maintaining the resulting SOA using information system. The choices usually adversely effect the cost and time and effort needed to achieve the goals managers typically seek in making the change to using SOA.

1. Categories of services granularity

In service-oriented architecture (SOA) software work, a service is instantiated as a piece of software that is or is to be reusable by the organization [7]. Depending upon their circumstances and management choices, some SOA-using organizations also require other characteristics, reflecting usually the operating environment in which the reusable software is to run. An example is running in a Web services environment [8]. Some of the other common characteristics are noted in the bullet sub-sections below.

The granularity of a service as used is determined by the dominance of the views and preferences of one of three groups of personnel in a SOA-using organization [22]. Middle and upper-level managers are especially attuned to giving service to whomever they regard as their customers. Their view of a service is something that contributes positively to serving whomever they regard as their customers. The users of the software-implemented systems in an organization often view a service more narrowly. A service is something that helps them get their assigned job done faster or more easily or better. The information systems (IS) and information technology (IT) group in an organization regard a service as software that directs the hardware in doing some wanted processing of data. The personnel specifying what processing of data is wanted may come from any of these same three groups.

Services often consist of smaller services, and sometimes may overlap other services. One basic and sometimes convenient unit of measure for the size of a service is the number of lines of source code or its equivalent needed for the implementation of the service. Since common kinds of software units have typical size ranges, those generic unit names are used in this paper as an indicator of the size needed for implementing a service [19]. Sixteen such categories of service granularity in SOA are listed and briefly sketched below, with the groups' interest in them noted with respect to implementing SOA. The order of listing is from smallest size to largest size (or as levels from lowest to highest):

Clones. Clones are more than two consecutive lines of imperative or declarative source code in any mix that are identical except for some or all of the operand names [5]. Clones may be executed either by drop in and drop out, or by transfers of control. The minimum size for a clone is three lines of source code, and the maximum commonly is less than 150 lines. When used as services, clones are useful for IS/IT personnel for their reusability, but not useful for users or managers.

Subroutines. Subroutines are named parts of routines that are executed by transfers of control, and usually consist of less than 250 lines of source code. When used as services, subroutines are like routines.

Modules. Modules are specialized routines designed and implemented to perform as integral parts of some larger unit of software. When used as services, modules are like routines.

Routines. Routines are named, and may have two or more constituent subroutines, or none. Routines are executed by transfers of control, and usually consist of less than 450 lines of source code. When used as services, routines are useful for IS/IT personnel and sometimes for users, but rarely for managers.

Mobile agents. Mobile agents are specialized routines that are executed remotely from the environment from which they are sent [1]. In some forms of SOA, services may be required to be qualified to perform as mobile agents, and then are useful for all three groups of personnel.

Procedures. Procedures are named aggregations of interacting routines, and consist usually of less than 1000 lines of source code. When used as services, they are more useful for IS/IT personnel than for users or managers.

Object methods. In object-oriented software work, objects normally have both data and methods that use those and other data. Object methods usually are examples of either routines or procedures, and as services, are treated as such.

Components. Components usually are procedures or modules or methods used by an organization as a part of its portfolio of software, but provided by a third party, such as a vendor or an open source [16]. Maintenance of a component is normally done by its provider. Hence, when used as services, components are treated like routines.

Files. Some components, procedures, modules, or objects with their methods are packaged and then termed "files." The data in such files may be compiled object code ("binary"), or source code, or both. When used as services, such files are treated by the three groups as they would treat the respective contents of the files.

Mashups. Some components, procedures, or objects with their methods are loosely lumped together based usually on some sharing of data, and termed "mashups" [23]. When used as services, mashups are usually treated like routines, or more rarely like programs.

Aspects. Aspects are concern-focused clusterings of source code that execute when triggered by the needs of other pieces of software [17]. Aspects may be routines, modules, procedures, components, or files, depending upon their form. When used as services, they are usually treated by the three groups based upon the form of the aspect.

Programs. Programs are named interacting aggregates of routines, components, procedures, objects, and aspects, that serve as parts of one or more subsystems. When used as services, programs are useful for IS/IT personnel, and are used extensively by users.

Subsystems. Subsystems are named interacting aggregates of programs, although sometimes a single program may be also a subsystem or a system. When used as services, subsystems are usually used like programs by users, but are useful also to IS/IT personnel and to a lesser extent to managers.

Applications. Applications are named subsystems or systems. As services, they are primarily useful to users and to managers. Managers sometimes use the term "application" to refer to any large piece of software, such as a program, subsystem, or system.

SaaS. Software as a service (SaaS) provides user access to application software [2]. Selected applications can then be used (run or executed) by an organization at the cost of a usage charge, and the cost of transmitting data to and from the SaaS Internet site of the organization that has the application software to be executed. When used as services, SaaS applications can be useful for personnel in all three groups.

Systems. Systems are named interacting aggregates of subsystems that may exceed one million lines of source code, but sometimes may consist of only a single program. When used as services, systems are most useful to managers. Users of systems usually make use of only portions of the systems. For example, the user personnel of an accounting system usually specialize and use only the included applications helpful to them, such as payroll or accounts receivable. And for example, IS/IT personnel only occasionally work with an accounting system as a whole, but often work with its parts.

2. Effects of granularity

2.1. Concepts and communication

The original specification of what is to be a service was rapidly lost sight of as the SOA bandwagon attracted more attention and local customization was de facto encouraged of what is SOA [8, 11]. In hindsight, the loss of the original view of SOA appears to have arisen from the different perspectives and interests of the IS/IT personnel, of the users of computer-implemented systems, and of the managers in organizations. Within each group, gradations in perspectives and interests are also present. For example, in the manager group, top-level (such as "C-level" and higher) personnel typically are more focused on organization wide matters than are area, department, section, or unit level managers.

Of the three groups, the user group is usually the least concerned about what is a service. As long as the users are getting substantially the performance they want from the applications in the organization's portfolio of information systems and accessed outside resources, and are not being directly charged on the basis of their usage, they could not care less about whether SOA is there or not. The other two groups are more likely to have concerns about what is a service.

The manager group typically considers a service to be at the system or at least subsystem level, and when used as part of a SOA implementation, as a potential means for progress toward attractive goals [4, 6 Chapter 3]. The IS/IT group typically considers a service to be at the routine or subroutine level, and when used as part of a SOA implementation, as another piece of software to be maintained. Hence, an attempted communication about services for or in or re SOA among members any two or three of the groups usually results in miscommunication to some degree, and sometimes in hype. Attempts to bridge the resulting communication gaps have taken several forms, including using local redefinitions of what is meant by the term " SOA."

2.2. IS/IT attempts to meet managers' expectations

IS/IT personnel have generally thus far not succeeded in getting either manager or user acceptance for what they consider as services in a SOA context. Instead, managers

have usually assigned IS/IT personnel to implement SOA as the managers envision SOA by reusing as much of the existing software as possible and doing as little additional software maintenance as possible.

To attempt to do this, the IS/IT response has usually relied on inserting the use of wrappers or middleware or enterprise service buses (ESB), or some combination of them. IS/IT and vendor proposals to make such insertions have been justified to managers generally on the grounds that they are software development actions (not maintenance) that can bring faster adaptability of the organization to meet the ever changing needs of the user group. Such proposals has also been justified as avoiding having to do adaptive software maintenance work to upgrade some existing software in sizes ranging from clones through objects, in order to change that existing software to make it work as part of SOA.

Wrappers are software that give the appearance of a different interface for existing pieces of software than its usual or former interface, while minimizing doing adaptive maintenance on the existing software [21]. With a wrapper, relevant invocations, input, and output of the existing piece of software are automatically routed through the wrapper, and any needed data conversions done. The usual reason for creating wrappers is to facilitate the reusability of existing software. Another is to provide an XML interface for existing software, because XML interfaces are required in some versions of SOA, but currently are rare in existing software.

Middleware is software that provides interoperability among pieces of existing software to assist their use as part of a SOA implementation, or provides additional services if needed in a SOA implementation [14, 20]. Since one of the lures of SOA for managers is the claim that it facilitates developing new applications quickly to meet users' expressed needs, managers sometimes warmly greet additional services via middleware. More commonly, however, managers find that the process of implementing SOA gets bogged down in delays in getting adequate interoperability, and managers reluctantly find themselves approving more middleware development, or the acquisition of middleware from various vendor's offerings.

Where incorporating existing (and especially "legacy") software is a major portion of a SOA implementation, one variety of middleware often gets included. It is the enterprise service bus (ESB) [15]. It works as an integrating communication means, and as a router for some execution requests. Also it can work as both a router and buffer for data flows among the parts of the SOA implementation. This can reduce the need for using some wrappers, since the buffering may include data conversions, such as to or from XML.

2.3. Distributed execution environment

Meeting managers' directives to attempt implementing their vision of SOA encounters a cost-raising constraint. That constraint is the growing role of the Internet in the transactions involving the organization, and the normal built-in recognition of the role of the Internet and its World Wide Web in SOA. A consequence of the constraint is that a SOA implementation almost always has to be able to operate in a distributed environment [4, 6 Chapter 3]. In nearly all organizations this means at a minimum working with the Internet and geographically distributed users, and for some organizations at a maximum meeting those organizations' needs from globally distributed computer sites [3].

As a part of its normal performance, any working implementation of SOA has to be able to receive requests and remote procedure calls coming in via the Internet, and to send results and remote procedure calls out via the Internet. The software comprising the entire SOA implementation need not be at just one physical location (site), but portions of the SOA software may exist and be working asynchronously at different and/or multiple sites. For coordinated action among the distributed sites, SOA software usually uses the Internet.

The tie of SOA to a distributed execution environment has been a source of concern to some managers of users when the tie has consequentially also involved or encouraged a distributed data environment. Some managers of users regard having control or ownership of data as being a key foundation for their effectiveness in their user manager roles [6 Chapter 6]. Such managers usually have sought and often won more local customizations in SOA implementations to preserve restricted access to and distribution of selected parts of an organization's data, usually via some database modifications and/or some objects' data and methods.

3. Issues related to service granularity

3.1. Organizational distinctiveness and culture

Commonly, an organization distinguishes itself from other organizations by encouraging its own "culture." In practice that means doing things a little differently from how other organizations do things, and that can include how the information systems interface with the employees. The bias toward culture differences can affect the implementation of a SOA. One version of SOA does not fit all organizations equally well.

Managers usually take culture into account when they consider introducing or extending SOA in an organization, for they want the users to be facilitated in doing their work "our way." To the extent managers give it consideration, this makes the user group be customers of services meriting quality of service (QoS) care from the information systems. This gives direction to the services that the managers expect to find in or have put into a SOA implementation.

Customization is one common way to add to the distinctiveness of the information systems running as part of

a SOA implementation. Modifying open source software or an organization's own reusable software adds maintenance effort to creating a SOA implementation, and negates some of the major advantages of using reusable and open source software in a SOA implementation. Acquiring vendor-supplied proprietary software may limit the extent of the customization possible. But if the proprietary software is not popular, then its unpopularity may obviate the need to do any major customization. Any customization may also have an adverse effect on reliability or security.

3.2. Reliability and security

Users tend to take reliability and security in information systems for granted, something that is taken care of for them. Managers tend to be more aware of the need for reliability in information systems' performance, than in their security. Yet security breaches can result in violations of reliability.

Unfortunately, SOA provides an environment that is comfortable for hackers and intruders for three major reasons. One is SOA's close ties with the Internet. This makes implemented SOA software be a potential target for denial of service attacks, for example. Hence, the common defenses associated with Internet usage, such as firewalls, are very important for use with SOA. A second reason is the big use of remote procedure calls in most SOA implementations, because remote procedure calls can be hijacked and infected. Defending against corrupted calls often becomes part of the tasks assigned to middleware and wrappers. A third reason arises from the distributed environment provided by SOA implementations. Infections can be spread easily among the distributed sites, because the usual Internet traffic between parts of the SOA implementation usually gets light inspection by the firewalls.

3.3. SLA and alignment

Because of the complexity added by wrappers and middleware to the complexity arising from SOA's close ties with the Internet, users tend to worry about the robustness and reliability of SOA implementations. The classic line of defense for users to protect themselves from too frequent service outages or other service impairments, has been to ask for service level agreements (SLAs) from the service provider [12]. Such requests typically result in negotiations between the middle managers in the user and IS/IT groups. The availability of good quantitative metrics makes SLA use both more effective and less contentious.

While the manager group is rarely involved with SLAs, the personnel of the manager group are involved with the effects of SOA on the alignment of IS/IT with their general goals and objectives for the organization overall. SOA implementation software projects typically start small, and then spread out being more inclusive. Hence, managers often get concerned about their respective domains being either penalized or favored as the SOA work progresses, and depending on how well it progresses. This can result in political conflicts and power plays within an organization.

3.4. Maintainability

Keeping ahead of the hackers and intruders requires a continuing and varied software maintenance effort. For SOA implementations, the effort usually is a mix of three types of maintenance: enhancive, adaptive, and preventative. In many organization, this maintenance work is done only partially by the regular IS/IT personnel, and done mostly by security personnel who may or may not be assigned as IT personnel.

The more significant burden of software maintenance comes from the way that the SOA has been implemented. If vendor-provided or open-source software serves as most of the middleware, then applying supplier-provided patches and updates is an important part of the maintenance. Some of that work has to be done dynamically on-the-fly without turning off the execution of the software being updated.

If in-house wrappers and middleware have been used instead of vendor or open-source software, then the maintenance may be less costly because of the trade-off between the added costs of employees and the added costs maintenance fees. Having to apply some changes on-the-fly does not disappear, but the personnel hours expended depends on the relative amounts of the various levels of granularity used in the SOA implementation.

The informal field experience thus far is that the higher the proportion of implementing from the higher levels of granularity, the higher is the subsequent stream of maintenance expenses. The situation is the common economics trade-off of trying for the optimum balance between the up front investment in maintenance and the continuing stream of subsequent maintenance costs. Neither can be held to zero money outlay. The underlying reason is the additional complexity introduced by trying to make existing software appear to do what it was not designed and implemented originally to do. Another reason is the complexity introduced by changing data flows and control sequences, as by such techniques as inserting wrappers and middleware. Increased complexity results usually in decreased maintainability.

As was noted at a conference in Europe last year, one potentially powerful way to make SOA software more maintainable would be to provide IS/IT personnel with a high-level vocabulary of service commands [9]. Those commands could the be used instead of IS/IT's current vocabulary of commands from the newer and most powerful design or programming languages, such as those of executable UML. The commands, however, to be useful would have to be supported by compilers or translators to convert them into executable object code. Currently, we have neither the commands nor the supporting compilers or translators.

3.5. Hardware considerations

Today and in at least the near-term future, organizations will continue to operate by using data to produce a compact communicable manipulable model of their operations and activities. Thus far, computers have greatly reduced the cost and greatly increased the power and usefulness of such data models. Software has enabled using the {data in :: data out} normal capability of computers to handle a tremendous diversity of data models in an enormous diversity of fields, such as music, bioinformatics, taxation, genealogy, etc. To use computers, we have to express as data the things to be conceptually worked with. Unfortunately, we have not yet succeeded in cleanly expressing "service" as data. But to use today computers in SOA, we have to work with what we have. To do that today, we have to settle at least temporarily upon some way of expressing a service as data. Here again, the granularity choice has major effects.

The lower levels of granularity can far more easily and cleanly be used as services, since the complexity level of them individually is relatively low. IS/IT personnel can usually and fairly easily also express these as functions in a strict-use mathematical sense. And IS/IT personnel know how to use function-rigorous techniques to create subroutines and routines, and then use those to build instances of almost any of the higher levels of service granularity [18]. To implement something successfully on a computer requires a full accurate expression of the software parts and their interactions in specific data terms.

Such a building process while possible is not fast or easy, and hence is relatively costly. To date, the economics of implementing SOA from the bottom starting with the lower levels of granularity has appeared unattractive to the manager group in organizations. Thus far, the track record of the compromises and high-level granularity approved by organizations' managers as courses of action to be attempted in planing and implementing SOA has been littered with cost and time overruns, shortfalls from targeted capability, and some recognized failures [13].

3.6. Software considerations

Largely because of the software factors, the financial track record thus far for SOA implementations has been mostly disappointing to the manager group. Part of this may flow from the usual track record of development projects, for development projects are generally riskier than and with higher failure rates and lower user satisfaction rates than maintenance projects [10]. Following the manager groups' choices of working from the higher levels of service granularity with minimal maintenance changes to the existing (including legacy) software, runs counter to a common IS/IT rule of thumb: getting software to act as it was not designed and implemented to perform increases complexity and raises the cost of future work done on that software. Adding to that complexity increase is that the distributed asynchronous execution obscures traceability and governance, reduces the effectiveness of controls, and makes testing more difficult. The complexity increase and upward cost bias are impairing progress toward a common manager goal in going to SOA, the goal of enabling fielding faster and at a lower cost the new applications that users need, and that could boost a manager's standing in an organization. To possibly mitigate that impairment, the IS/IT group would like, but as noted earlier does not yet have, a new methodology to enable designing and implementing software in terms of services [9].

Using the Internet in working with customers, suppliers, and an organization's own distributed locations continues to grow in importance because of the opportunities for growth that it offers. In practice, exploiting that opportunity without effective countermeasures in place and working, exposes the assets of an organization to increasingly sophisticated criminally-motivated attacks via the Internet.

4. Conclusions

Managers' typical choices thus far have been to have SOA be implemented as development projects nominally reusing existing software at the higher levels of granularity but in distributed environments open to the Internet. The managers have been generally accepting of the commonly higher failure rates of development projects compared to maintenance projects. Concurrently, the managers have discouraged software maintenance work on the existing software, but tolerated developing or acquiring and using middleware and wrappers. The results of these choices has been projects often troubled by overruns on cost and schedule and shortfalls on meeting project expectations, while increasing the complexity of their organizations' software assets. Unless offset, that added complexity will in turn increase the difficulty and cost of maintaining user satisfaction with the SOA software.

5. References

1. Bernichi, M. and Mourlin, F., 2007, "Software management based on mobile agents," *Proceedings Second International Conference on Systems and Network Communications (ICSNC 2007)*, IEEE Computer Society, Los Alamitos CA, pp. 64–69.
2. Choudhary, V., 2007, "Software as a Service: implications for investment in software development," *Proceedings 40th Annual Hawaii Internation Conference on System Sciences (HICSS'07)*, IEEE Computer Society, Los Alamitos CA, pp. 209–218.
3. Cuomo, G., 2005, "IBM SOA 'on the Edge,'" *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, ACM Press, New York NY, pp. 840–843.
4. Dan, A.; Johnson, R.; and Arsanjani, A., 2007, "Information as a service: modeling and realization," *Proceed-*

ings International Workshop on Systems Development in SOA Environments (SDSOA'07), IEEE Computer Society Press, Los Alamitos CA, pp. 2–7.

5. Ducasse, S.; Nierstrasz, O.; and Rieger, M., 2006, "On the effectiveness of clone detection by string matching," *Journal of Software Maintenance and Evolution*, Vol. 18, No. 1, pp. 37–58.
6. Erl, T., 2008, *SOA Principles of Service Design*, Prentice Hall, Upper Saddle River NJ, 573 pp.
7. Erradi, A., 2006, "SOAF: an architectural framework for service definition and realization," *Proceedings International Conference on Services Computing (SCC'06)*, IEEE Computer Society, Los Alamitos CA, pp. 151–158.
8. Ferris, C, and Farrell, J., 2001, "What are Web services?" *Communications of the ACM*, Vol. 46, No. 6, p. 31.
9. ICSM. 2007, "MESOA working session on maintenance and evolution of service-oriented systems," *Proceedings of the 2007 IEEE International Conference on Software Maintenance*, IEEE Computer Society, Los Alamitos CA, Appendix. URL of MESOA wiki is pending.
10. Jones, C., 2003, "Variations in software development practices," *IEEE Software*, Vol. 20, No. 6, pp. 22–27.
11. Jones, S., 2005, "Toward an acceptable definition of service," *IEEE Software*, Vol. 22, No. 3, pp. 87–93.
12. Kajko-Mattsson, M.; Ahnlund, C.; and Lundberg, E., 2004, "CM^3: service level agreements," *Proceedings 20th IEEE International Conference on Software Maintenance*, IEEE Computer Society, Los Alamitos CA, pp. 432–436.
13. Kajko-Mattsson, M.; Lewis, G. A.; and Smith, D. B., 2007, "A framework for roles for development, evolution and maintenance of SOA-based systems," *Proceedings International Workshop on Systems Development in SOA Environments (SDSOA'07)*, IEEE Computer Society, Los Alamitos CA, pp. 7–12.
14. Kon, F.; Costa, F.; Blair, G.; and Campbell, R. H., 2002, "The case for reflective middleware," *Communications of the ACM*, Vol. 45, No. 6, pp. 33–38.
15. Manes, A. T., 2007, *Enterprise Service Bus: A Definition*, Burton Group, Midvale UT, 35 pp.
16. Mariani, L., and Pezzè, M., 2007, "Dynamic detection of COTS component incompatibility," *IEEE Software*, Vol. 24, No. 5, pp. 76–85.
17. Pace, J. A. D., and Campo, M. R., 2001, "Analyzing the role of aspects in software design," *Communications of the ACM*, Vol. 44, No. 10, pp. 67–73.
18. Prowell, S. J.; Trammell, C. J.; Linger, R. C.; and Poore, J. H., 1999, *Cleanroom Software Engineering*, Addison Wesley, Reading MA, 390 pp.
19. Ralston, A., and Reilly, E. D. (Editors), 1993, *Encyclopedia of Computer Science, Third Edition*, Van Nostrand Reinhold, New York NY, 1558 pp. Refer to article with size item name (e.g., routine).
20. Sarna-Starosta, B.; Stirewalt, R. E. K.; and Dillon, L. K., 2007, "Contracts and middleware for safe SOA applications," *Proceedings International Workshop on Systems Development in SOA Environments (SDSOA'07)*, IEEE Computer Society, Los Alamitos CA, pp. 5–10.
21. Sneed, H. M., 2001, "Wrapping legacy COBOL programs," *Proceedings Eighth Working Conference on Reverse Engineering (WCRE 2001)*, IEEE Computer Society, Los Alamitos CA, pp. 189–197.
22. Sneed, H. M., 2006, "Integrating legacy software into a service oriented architecture," *Proceedings of the Conference on Software Maintenance and Reengineering (CSMR'06)*, IEEE Computer Society, Los Alamitos CA, pp. 3–14.
23. Taft, D. K., 2008, "Mashups spawn debate," *eWeek*, Vol. 25, No. 6, p. 20.

Securing Service-Oriented Systems Using State-Based XML Firewall*

Abhinay Reddyreddy and Haiping Xu
Computer and Information Science Department
University of Massachusetts Dartmouth
North Dartmouth, MA 02747, USA
{g_areddyreddy, hxu}@umassd.edu

Abstract. *Web services security has been a challenging issue in recent years because current security mechanisms, such as conventional firewalls, are not sufficient for protecting service-oriented systems from XML-based attacks. In order to provide effective security mechanisms for service-oriented systems, XML firewalls were recently introduced as an extension to conventional firewalls for web services security. In this paper, we present a state-based XML firewall architecture that supports role-based access control and detection of XML-based attacks. We develop a detailed design of our state-based XML firewall by defining state-based information, user information, and various access control policies and detection rules. The detection rules are modularized into separate units, which support real-time detection and verification of various types of XML-based attacks using state-based information and user information. To illustrate the effectiveness of our approach, we develop a prototype state-based XML firewall, and demonstrate how XML-based attacks can be efficiently detected.*

1. Introduction

Enterprises are increasingly employing web services technology in order to achieve interoperability at application levels. Web services are both platform and language independent service components that can be exposed using a standard Web Services Description Language (WSDL) and registered at UDDI registries. They can be automatically discovered over the Internet by potential clients, and support loosely coupled interactions between applications through a standard XML-based protocol, called Simple Object Access Protocol (SOAP). Since web services create open interfaces into core enterprise applications and data, attacks on web services can be more severe than those attacks perpetrated via e-mail, web servers and network connections.

* This material is based upon work supported by the Chancellor's Research Fund and UMass Joseph P. Healey Endowment Grants, and the Research Seed Initiative Fund (RSIF), College of Engineering, UMass Dartmouth.

Conventional firewalls are not sufficient for protecting service-oriented systems because web services attackers can initiate attacks as request/response traffics using HTTP protocol that can pass conventional firewalls. The most commonly used conventional firewalls are package filtering firewalls, stateful inspection firewalls, and application level firewalls [1]. A packet filtering firewall only restricts IP addresses or TCP ports recorded in an IP table; however, the port 80 reserved for HTTP and SOAP traffics cannot be blocked on a server that hosts the web services. Thus, a malicious web service invocation can easily pass a packet filtering firewall. On the other hand, a stateful inspection firewall can keep track of TCP/IP connection states and take actions accordingly, but it does not look into packet contents. Similarly, an application level firewall also blocks only those suspicious network traffics with protocols that might be used by an attacker. For example, an application gateway for an FTP server can be configured to accept FTP traffics only and reject all packets using other protocols. Therefore, both stateful inspection firewalls and application level firewalls are not capable of detecting XML-based attacks, e.g., SQL injection attack and overloaded payload attack, which are embedded in XML-based messages [2, 3].

Lack of effective security mechanisms for web services is one of the major reasons why there are so many organizations hesitating to adopt service-oriented technologies despite their significant advantages. In this paper, we introduce an approach to securing service-oriented systems by developing a state-based XML firewall at the application level. Our approach supports Role-Based Access Control (RBAC) [4] for users and detection of XML-based attacks. The XML firewall design introduced in this paper is based on a formal XML firewall model presented in previous work [5], where access permissions to web services are only granted to users who are authenticated and authorized. We develop a detailed design by defining state-based information, user information, and various access control policies and detection rules. Finally, to demonstatrate the effectiveness of our approach, we implement a prototype state-based XML firewall for efficient detection of XML-based attacks to a hospital management service-oriented system.

2. Related Work

Web services security has been an active research area in recent years. Many organizations such as IBM and Cisco, tried to identify major threats to web services in order to protect service-oriented systems more effectively [6, 7]. Typical XML-based attacks include request flooding attack, SQL injection, parameter tampering, overloaded payload attack, and recursive payload attack [2]. A request flooding attack is a type of XML Denial of Service (XDoS) attack, where the attacker floods the web service provider with a large number of web service requests in order to exhaust the resources at the server side. XDoS attacks are similar to packet-based DoS attacks that flood servers with lots of data (e.g., SYN packets); however, conventional firewalls cannot detect XDoS attacks because XDoS attacks are threats to the availability of web services rather than network connections. An SQL injection attack involves tampering the input fields of database requests in order to obtain unauthorized access to data or stored procedures; while a parameter tampering attack is a process of tampering the method parameters passed to a web service operation, and resulting in undesired service behaviors. An overloaded payload attack and a recursive payload attack can exhaust the XML parser of a service provider by sending huge XML data and embedding deeply nested elements in a web service request, respectively.

There is very little previous work on protecting web service providers from being attacked. Fernandez proposed a pattern-based language for XML firewall [8]. Two patterns for design of XML firewall were proposed, which are security assertion coordination pattern using role-based access control for access to distributed resources, and filter pattern for filtering XML messages or documents according to institution policies. Hoktamp discussed the need for XML firewall and possible techniques to protect web services [9]. He analyzed the security issues at three levels of enterprise application integration, namely intranet, extranet and Internet. Cremonini et al. discussed about integrating XML firewall with existing web services security specifications [10]. They analyzed serious security risks in stateful SOAP protocols such as WS-Reliable Messaging, and presented some design guidelines to develop semantics-aware firewalls that can be integrated with the Web Service Architecture (WSA). Bebawy et al. discussed how to apply business specific rules in a centralized manner to develop a web services firewall, called Netdgy [11]. In their implementation, SOAP messages are removed from the transport layer and examined for attack detection, and then induced back into the OSI stack if the XML message is not corrupt. The Netdgy system only supports prevention of limited types of web services attacks such as buffer overflow and SOAP-based DoS attacks. Furthermore, it does not provide any access control mechanisms for users; instead, it supports authorization based on IP tables, which is in the same manner as a conventional packet filtering firewall where messages originating from certain IP address are either dropped or accepted according to a list of blocked IP addresses. Different from the Netdgy system, our effort is to develop a modularized XML firewall that is customizable and targeted for various types of XML-based attacks, thus our approach provides a more comprehensive solution to web services security.

3. Development of State-Based XML Firewall

3.1 State-Based XML Firewall

Based on the formal model of XML firewall we introduced previously [5], we design the state-based XML firewall as a software module with four functional components, namely client interface, RBAC processor, SOAP filter, and admin interface, which coordinate to protect the web services deployed on a web server. As shown in Figure 1, the four major components in an XML firewall are supported by two databases: *User_Info* database and *State_Info* database, which store user information and state-based information, respectively. The client interface module interacts with web service clients and is responsible for receiving requests and sending responses back to the service clients. The actual web services can be deployed either on the same or a different machine where the XML firewall is installed; however, they can only accept requests from a service client through a service proxy defined in the client interface module. As illustrated in Figure 1, each deployed web service (e.g., *WS*1) has a corresponding web service proxy (e.g., *WS*1*P*) defined in the client interface module. The client interface module provides exactly the same interface for web service invocation as the deployed web services, so it is transparent to the web service clients. A client can access an actual web service only after it successfully passes through the XML firewall because the service proxies are the only interface for web service invocations. Authentication and authorization are the major features of the XML firewall for providing user access control. These features ensure that only valid users are allowed to access services. The login block defined in the client interface module provides a basic mechanism for user authentication; while the RBAC processor is responsible for authorizing a user with predefined roles and access permissions. The RBAC processor can determine whether a client has appropriate permissions to access a web service. If a malicious user is detected for a lack of access permissions, any attempts to access the web service by that user will be denied, and the user will be forced to log out of the system. In order to provide a valid duration for a user to invoke web services, we first define the concept of *user session* as follows.

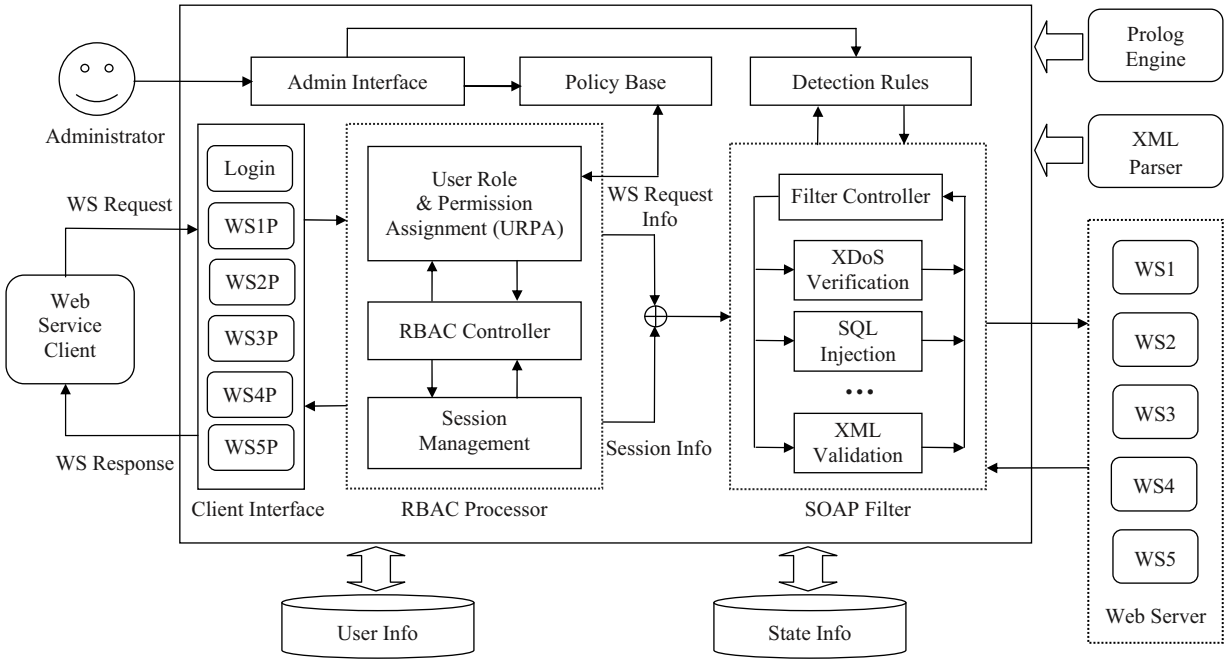


Figure 1. Architectural design of state-based XML firewall

Definition 3.1 A *user session* is defined as a 5-tuple *(UID, SID, RO, ST, ET)*, where *UID* is a user ID, *SID* is a session ID, *RO* is a set of roles that will be assigned to the user, *ST* is the session start time, and *ET* is the session expiration time. A user session is created when the user logs in and destroyed when the user logs out.

After a user logs in and passes the authentication step, his user information is transferred to the RBAC processor module for authorization. Before the User Role and Permission Assignment (URPA) module assigns the user appropriate roles and corresponding access permissions, the session management module in the RBAC processor creates a user session for that user, which has a start time and an expiration time. During the period of time when the session is valid, a user can make requests to web services without being authenticated again. The URPA module, which is used to assign roles to users and permissions to roles, interacts with the *Policy Base*, which is a repository of access control policies defined in Prolog by an administrator through the admin interface. The reasoning process for authorization is supported by a Prolog engine as well as user's information, such as a user's trust level, stored in *User_Info* database.

For every incoming web service request from a user, the RBAC controller verifies whether the associated user session is valid and the user has sufficient permissions to invoke the web service. If the user has enough permission to access the web service, his request in a form of XML message, along with the session information will be passed to the SOAP filter for threat detection and content analysis. Otherwise, the user's request will be denied by the RBAC controller immediately.

The filter controller in the SOAP filter module is responsible for detection of suspicious requests. It examines the session information passed to it as well as the data from the *User_Info* and *State_Info* databases to determine whether the user request is suspicious of any kind of attacks. The detection process is supported by the detection rules defined in Prolog by an administrator, which are modularized into different rule sets for detection of different types of XML-based attacks, e.g., an XDoS attack and an SQL injection attack. Thus, the modularized rule sets can be invoked individually, which support efficient reasoning in real-time. In addition, there is also a set of rules used by the filter controller for detection of attacker suspects. For example, when the filter controller detects a suspicious user with high frequency of requests (determined by predefined thresholds as shown in Section 3.4), the user's request will be passed to the XDoS verification module to verify if the user is performing an XDoS attack. Similarly, if the controller detects that a user request exceeds the normal packet size, the XML message will be sent to the XML validation module to verify for oversized payload attack. On the other hand, if a user request is a normal one, the request will be immediately passed to the web server for web service invocation.

The XDoS verification module requires investigation of a user's previous behavior in order to verify if a user is performing an XDoS attack. If the user has a very low trust level or has been suspected as an XDoS attacker for

a number of times, not only the request from that user will be dropped, but also the user's trust level may be degraded further. Different from the XDoS verification module, the SQL injection module only evaluates the parameters passed to a web service operation by matching them with predefined regular expressions in order to check for any malformed parameters or parameter tampering. Similarly, the XML validation module interacts with the XML parser to evaluate if the request message is well formed by comparing it with an XML schema, and also checks for the size and nesting depth of the message. If any malicious activity is detected and confirmed, the request for the web service invocation is denied immediately; otherwise, the request is passed to the web server for processing. When the service invocation is completed, the result will be forwarded back to the client through the client interface.

One of the major advantages of our approach is that the access policies and the detection rules are modularized; therefore, they can be dynamically updated without recompiling and reinstalling the XML firewall. As shown in Figure 1, a human administrator can add, remove or update any of the access policies and detection rules through the admin interface at runtime. However, during the updating process, the Prolog engine must wait until the updating process is completed.

3.2 Database Design for State-Based XML Firewall

In the detection process, the critical information used by the XML firewall for decision making is the data stored in *State_Info* and *User_Info* databases, which are used for detecting and verifying different types of XML-based attacks. In the following, we give some key definitions of data types used in *State_Info* and *User_Info* databases for detection of XDoS attacks.

Definition 3.2 A *user state* is a 5-tuple *(UID, SID, TR, FR, TL)*, where *UID* is the ID assigned to the user at the time of registration, *SID* is the ID of the session that is initiated, *TR* is the total number of requests made by the user in the current session, *RF* is the request frequency, i.e., the number of requests made by the user in a recent time interval, and *TL* is the user's current trust level.

Definition 3.3 A *firewall state* is a triple *(RE, DE, RT)*, where *RE* is the number of requests that are received by the XML firewall but not yet forwarded to the web server. *DE* is the number of requests that are being processed by the detection modules in the SOAP filter. *RT* is the number of requests in a recent time interval, e.g., the last five minutes. A firewall state is a measure of the work load on the XML firewall system.

Definition 3.4 A *web service state* is a triple *(WID, NR, SI)*, where *WID* is the ID of the web service, *NR* is the number of requests currently being processed by the web service, and *SI* is a state indication of the web service, which can be *busy*, *normal* or *free*. The state indication of a web service indicates the work load of the web service that is determined by thresholds set by an administrator.

Definition 3.5 A *user credential* is a 4-tuple *(UN, PW, UID, TL)*, where *UN* is the user name, *PW* is the password specified by the user at registration time, *UID* is the user ID, and *TL* is the current trust level assigned to the user. A user receives a "normal" trust level at the time of registration, and his trust level can be updated later at runtime based on the user's most recent behavior.

Based on the above state-based information and user information, the SOAP filter can detect and verify XDoS attacks in real-time. Note that the databases store not only the current state and user information, but also the previous states and the recent user information that are useful for attack verification.

3.3 Role-Based Access Control Policies

A role is an abstraction that represents a set of permissions that are needed to perform the tasks associated with a position. Role-based authorization policies specify the roles that each user may adopt, and the permissions associated with each role [4, 12]. From earlier research, it has been argued that it is desirable to separate policy from the application code, so policies can be easily changed over time [13]. Therefore, in this project, we choose Prolog as a specification language for both access control policies and detection rules. Prolog is a declarative language, and can be used to specify both facts and production rules or policies. With a solid mathematical foundation, Prolog allows to reason from a set of rules and supports meta-level reasoning, making policy conflict detection possible. Consider the following access control policies. In a hospital management system, a staff member (e.g., a billing clerk) and a pharmacist can only access a patient's contact and billing information but not his medical records. A patient can be assigned to a doctor or a nurse, who may have full access to the patient's medical records and contact information, but not his billing and account information. A patient can access all records of his own, including his contact information, billing and accounts, and medical records. The access control policies can be specified in Prolog as follows.

```
isValidRole(patient).
isValidRole(doctor). isValidRole(nurse).
isValidRole(staff). isValidRole(pharmacist).
assignRole(U,R) :- isValidRole(R).
canInvoke(R,T,billingService,accessBill):-
 contains(R,[staff,pharmacist,patient]),
 contains(T,[normal,high]).
canInvoke(R,T,billingService,computeBill):-
 contains(R,[staff,pharmacist]),
 contains(T,[normal,high]).
canInvoke(R,T,accessService,readRecord):-
 contains(R,[doctor,nurse,patient]),
```

```
 contains(T,[normal,high]).
canInvoke(R,T,accessService,writeRecord,P,U):-
 contains(R,[doctor,nurse]),
 contains(T,[normal,high]), assignPatient(P,U),
 assignRole(P,patient), assignRole(U,R).
canInvoke(R,T,contactService,accessContact):-
 contains(R,[staff,doctor,nurse,patient]),
 contains(T,[normal,high]).
```

Note that in the above Prolog code, *R* and *T* represent a user's role and the trust level of a user, respectively. Any user must take certain role and have at least a "normal" or "high" trust level before he can access any resource. The predicate *isValidRole* lists various roles defined in the system. The predicate *assignRole(U, R)* is true when a user with UID *U* is assigned a valid role *R*. Similarly, *assignPatient(P, U)* is true when the patient with UID *P* is assigned to a doctor or a nurse with UID *U*. The predicate *canInvoke* determines whether a user with a certain role has the permission to invoke a web service operation. For example, the predicate *canInvoke(R, T, accessService, readRecord)* specifies that a user with role *R* and trust level *T* can invoke the web service operation *readRecord* defined in web service *accessService*. Similarly, the predicate *canInvoke(R, T, accessService, writeRecord, P, U)* ensures that a doctor or a nurse *U* can update a patient *P*'s record only if the patient *P* has been assigned to the doctor or nurse with UID *U*.

3.4 Real-Time Detection of XML-Based Attacks

The SOAP filter is responsible for real-time detection of XML-based attacks. The process of detecting XML-based attacks involves two major steps, which are detection of suspicious SOAP messages and verification of attacks. Suspicious SOAP messages are detected by the filter controller, which uses the session and state information to find possible request flooding attacks, uses certain predefined patterns to find matched strings in the parameters passed to a web service operation; and also keeps track of the maximal allowed message size and the maximal allowed nesting depth in the incoming XML messages in order to detect oversized or recursive payload attacks. We now use an XDoS attack as an example to show how to detect XML-based attacks using our state-based XML firewall. To detect XDoS attacks, the filter controller looks into the session information to check if the current frequency of requests (e.g., the number of request during the last minute) made by a certain user exceeds the threshold set by an administrator. If the frequency exceeds the limit, any new requests from that user will be sent to the XDoS verification module for further analysis. Some sample rules used by the filter controller for XDoS detection are illustrated as follows.

```
checkThreshold(W,S,X):- threshold(W,SI,Y),X > Y.
threshold(accessService,busy,20).
threshold(accessService,normal,40).
threshold(accessService,free,60).
```

In the above rules, *W* is the web service name, *S* represents the session ID, and *X* is the number of requests per minute made by a user who is currently under investigation. The predicate *checkThreadhold* evaluates to true when the number of requests made by the user during the last minute exceeds the limit determined by the web service state indication. For this example, the state indication of a web service is *busy*, *normal*, or *free* if the number of requests processed by the web service during the last minute is larger than 40, between 20 and 40, or less than 20, respectively. According to the above rules, when the web service is *busy*, *normal* or *free*, the corresponding limit on number of requests per minute is 20, 40 or 60, respectively. Note that the information about the web service state and the number of requests the user made during the last minute are stored in *State_Info* database. To simplify matters, the threshold in our current XML firewall implementation does not depend on the firewall state that is specified in Definition 3.3.

If a query to the predicate *checkThreshold* returns true, the corresponding request will be passed to the XDoS verification module where the user's violation history is analyzed. The following Prolog rules demonstrate how to verify an attacker and when to degrade a user's trust level.

```
xdosVerify(U,T):- inspectHistory(U,T,V).
inspectHistory(U,T,V):-
  T = high, dataConnect(U,3,V), V = '3',
  degradeTrustLevel(U,normal).
inspectHistory(U,T,V):-
  T = normal, dataConnect(U,5,V), V = '3',
  degradeTrustLevel(U,low).
inspectHistory(U,T,V):-
  T = low, dataConnect(U,7,V), V = '3'.
  degradeTrustLevel(U,permanentlyBlocked)
dataConnect(U,X,V):-
  java_object('DataConnect',[],data),
  data<-getHistorySessionStatus(U,X) returns V.
degradeTrustLevel(U,T):-
  java_object('DataConnect',[],data),
  data <- recordTrustLevel(U,X).
```

The Prolog code inspects a user's violation history of exceeding service invocation frequency threshold. If the user's trust level is "high", the XDoS verification module only checks the user's previous 3 sessions. If the user has 3 violations, his trust level will be degraded to "normal". On the other hand, if the user's trust level is "normal" or "low", then the user's previous 5 or 7 sessions need to be checked. Similarly, when the user reaches the limit of 3 violations, his trust level will be degraded to "low" or "permanentlyBlocked", respectively. In all above cases, if a query to the predicate *xdosVerify* evaluates to true, the user's current session will be immediately closed. In this case, the user must log in again before he can make further requests. Note that the Prolog code listed above requires invoking Java methods *getHistorySessionStatus* and *recordTrustLevel* to acquire information from *State_Info* database, and record a user's trust level as history information in *User_Info* database, respectively.

Another example to show how to detect attacks using our XML firewall is to detect SQL injection attacks. SQL injection is a technique used to exploit the vulnerabilities in web applications that communicate with databases. The basic idea behind SQL injection is to convince the application to run some malicious SQL code that may result in unauthorized data access or data loss. SQL injection attacks mostly occur due to a lack of user input validation. Although SQL injection is a general technique to attack web-based applications, in the context of service-oriented systems, it can tamper web service parameters which are embedded in XML messages. Thus, in this paper, we treat it as a type of XML-based attack. A simple example of SQL injection attack is called concatenated query attack, where the user manipulates a parameter to form a concatenated query. When a normal query "`SELETE * FROM users WHERE userid = 'user1'`" is manipulated to "`SELECT * FROM users WHERE userid = 'user1'; DELETE FROM users; -- x'`", the execution of the query results in data loss.

In our current implementation of XML firewall, the SOAP filter uses regular expressions to specify string patterns such as concatenation of "`';`" and "`';--`". If any input string matches one of the predefined patterns, the user will be detected as an attacker for SQL injection, and the user's current session will be closed immediately.

4. A Case Study

In this section, we use a case study to demonstrate how a state-based XML firewall can be used to effectively detect XML-based attacks. We developed a prototype XML firewall, and installed it on the same machine where a service-oriented system was deployed. The service-oriented system we adopted in this case study is a hospital management system, where different roles and access control policies are defined to determine a user's access permission to specific services. The hospital management system is an adaptation of the system presented in previous work [13], which is implemented as a service-oriented system. The related user roles as well as their corresponding access permissions are the same as those defined in Section 3.3. We now first simulate an SQL injection attack by accessing the web service *accessService*, which allows a user with sufficient permissions to write medical records for a patient. Consider *User1* with a patient role who is assigned to nurse *User2*. Since *User1* is assigned to *User2*, *User2* has permission to write *User1*'s medical records by invoking *writeRecord* operation defined in web service *accessService*. The invocation requires four parameters, namely the ID of the user who writes the record, the ID of the patient whose record is to be updated, a string containing medical report information, and the type of the report. A legitimate request from the nurse could be *writeRecord("User2", "User1", "The patient reacted abnormally to new drugs.", "Observation")*, which results in an SQL query as follows.

```
INSERT INTO patientRecords VALUES('User2',
'User1', 'The patient reacted abnormally to new
drugs.','Observation');
```

Now a malicious user may perform an SQL injection attack by tampering the parameters in the web service invocation. *User2* may send the fourth parameter as "`Observation'); DELETE FROM users; -- dummystring`". The resultant query in the web service will delete all the records in *users* table if the server allows execution of multiple queries.

With the installed XML firewall, when *User2* makes such a request, the XML firewall can successfully detect the SQL injection attack and prevents unauthorized data access by checking the parameters of the request against predefined regular expressions. Figure 2 is a snapshot of the log information showing the successful detection of a simulated SQL injection attack.

Figure 2. Log information for SQL injection detection

To demonstrate that our prototype XML firewall can effectively detect and prevent XDoS attacks, we simulate request flooding attacks on a web service with a large number of requests from an attacker, and record the response behavior of the server for requests from a normal user. We choose the report generation service implemented in the service-oriented hospital management system because it consumes significant amount of memory space and CPU time. The web service takes around 10 seconds to process a request; thus, the normal response time should be around 10 seconds. We now set up the flooding attack with a number of threads, each of which sends web service requests continuously to the report generation service. When the XML firewall was disabled, we observed that when the number of requests received by the server increases, the response time of a request from a normal user increases significantly. When the frequency of requests reaches around 128 per minute, the web service becomes unavailable to the normal user because the server crashes due to a heap space error. This

is illustrated by one of the curve (denoted as "without XML Firewall") in Figure 3. When we enable our XML firewall and set an appropriate threshold, the web server can be successfully prevented from crashing. Figure 3 shows two other curves that represent the experimental results with the XML firewall enabled when the thresholds for the firewall with *free* state indication are set to 80 and 60, respectively. As shown in Figure 3, when the threshold is 80, the worst response time is 25 seconds, but it drops to normal response time when the attacker increases the request frequency further. To enhance performance, we lower the threshold from 80 to 60, and the worst response time now becomes 17 seconds.

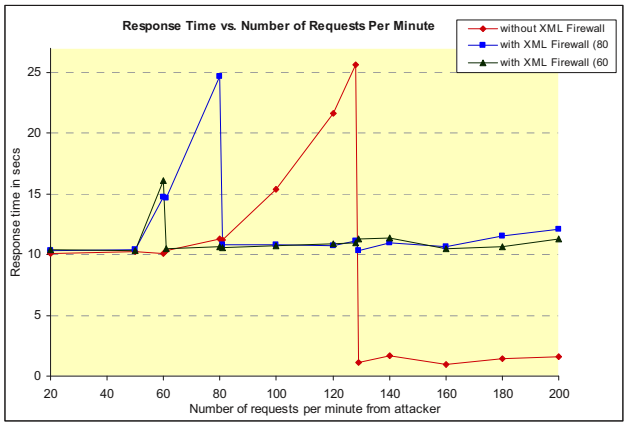


Figure 3. Experimental results for XDoS attacks

Note that a very high threshold could overload the system while a very low threshold might block legitimate users with high request rates. Thus, it is important for the administrator to choose an appropriate threshold for the XML firewall in order to make it work efficiently.

5. Conclusions and Future Work

Service-oriented systems are increasingly deployed over the Internet due to their standardized protocols and techniques that enable the efficient integration of loosely coupled applications over networks. However, due to the open interface for service-oriented architecture, attacks on service-oriented systems are more complicated than traditional attacks that can be handled by conventional firewalls. Thus, there is a pressing need to introduce new security mechanisms to protect service-oriented systems. In this paper, we introduced a state-based XML firewall, which can be used to protect service provider from various XML-based attacks. We developed a detailed design of our state-based XML firewall, and implemented a prototype XML firewall. Our experimental results show that our prototype XML firewall can effectively protect web services from various XML-based attacks. In our future work, we will study new types of XML-based attacks and show how their corresponding attack verification modules can be easily integrated into our current implemented system due to the modular design. We will also consider adopting agent-based technology to provide more intelligence in XML firewall for efficient detection and verification of XML-based attacks.

References

[1] E. B. Fernandez, M. M. Larrondo-Petrie, N. Seliya, N. Delessy-Gassant, and M. Schumacher, "A Pattern Language for Firewalls," In M. Schumacher, *et al.* (Eds.), *Security Patterns: Integrating Security and Systems Engineering*, Wiley, March 2006.

[2] E. Moradian and A. Håkansson, "Possible Attacks on XML Web Services," *International Journal of Computer Science and Network Security (IJCSNS)*, Vol.6, No.1B, January 2006, pp. 154-170.

[3] M. Andrews and J. A. Whittaker, *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*, Addison-Wesley Professional, February 2006.

[4] H. Feinstein, R. Sandhu, E. Coyne, and C. Youman, "Role-Based Access Control Models," *IEEE Computer*, Vol. 29, No. 2, 1996, pp. 38-47.

[5] H. Xu, M. Ayachit and A. Reddyreddy, "Formal Modeling and Analysis of XML Firewall for Service Oriented Systems," *International Journal of Security and Networks (IJSN)*, Vol. 3, No. 3, 2008.

[6] P. Crocker and B. Thompson, "Integrating WebSphere DataPower SOA Appliances with WebSphere MQ," *Technical Report*, IBM Hursley Software Lab, March 2007.

[7] Reactivity, "Architecting the Infrastructure for SOA and XML," *White Paper*, Cisco Systems, Inc. 2007.

[8] E. B. Fernandez, "Two Patterns for Web Services Security," In *Proceedings of the 2004 International Symposium on Web Services and Applications (ISWS'04)*, Las Vegas, Nevada, 2004.

[9] M. Holtkamp, "The Role of XML Firewalls for Web Services," *The 1st Twente Student Conference on IT*, Track B, June 2004.

[10] M. Cremonini, S. Vimercati, E. Damiani, and P. Samarati, "An XML-Based Approach to Combine Firewalls and Web Services Security Specifications", In *Proceedings of the 2003 ACM Workshop on XML Security*, Fairfax, Virginia, 2003, pp. 69-78.

[11] R. Bebawy, H. Sabry, S. El-Kassas, Y. Hanna, and Y. Youssef, "Nedgty: Web Services Firewall," In *Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, 2005, pp. 597-601.

[12] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Transactions on Information and System Security (TISSEC)*, Vol. 4, No. 3, August 2001, pp. 224-274.

[13] M. Y. Becker and P. Sewell, "Cassandra: Flexible Trust Management, Applied to Electronic Health Records," In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, 2004, pp. 139-154.

Toward Model Checking Web Services over the Web

John C. Sloan
Taghi M. Khoshgoftaar*

Abstract

Due to their high level of abstraction, web services blur the traditional distinction between model *and* code*. Consequently, executable artifacts can now be written to a level of abstraction that more closely approximates that of models used for model checking. Unlike traditional localized white-box software artifacts, the distributed black-box nature of web services demand exhaustive verification. Recent usage trends for model checkers coupled with the rise of web services, suggest a different approach to this computationally intensive form of verification. This paper outlines these trends and describes a web-enabled approach to verification of web services. We outline how activities associated with model checking may be migrated into a web services framework, enabling practitioners to more easily incorporate model checking into their solutions.*

Keywords: web services, model checking, service oriented architecture, formal verification.

1. Introduction

As a composition of loosely coupled autonomous services, each having a known interface, advertised functionality, and specified behavior, Service Oriented Architectures (SOA) are most commonly implemented as web services. Among other benefits, SOA's free developers from concerns over platform, implementation, and versioning. These freedoms, however, render most traditional testing techniques ineffective. Without access to source code of services that may not behave as advertised, unforeseen usage scenarios and implicit assumptions can cause race conditions that end in deadlock or other undesired interaction. In this setting, management would be reluctant to deploy safety or fiscally-critical applications as web services.

To remedy this, Nakajima [18] applied model checking to compositions of web services. Given glue code artifacts written to a predecessor of Web Services Business Process Execution Language (WS-BPEL) and Web Services Description Language (WSDL), Nakajima proposed converting these artifacts to a finite state model suitable for model checking. Model checking is a technique for exhaustively verifying compositions by automatically enumerating every usage scenario, checking each for violations of specified properties. More precisely, given a finite state model of a system, model checking is a means of verifying if certain properties hold for reachable states within the state space generated from that model. A survey of formal methods, including model checking, appears in [6].

We describe how model checkers have recently been used, from which we delineate features of a proposed SOA for model checking web services, and suggest one such architecture. This work will facilitate development of best-of-breed verification environments by embedding model checking techniques into web-enabled applications.

The remainder of this paper is organized as follows: Section 2 describes how model checkers have been recently used. Based on these usage trends we describe improved means of generating checkable models from process definitions in Section 3, followed by a top-level architecture in Section 4 concluding with Section 5.

2. Advanced Usage Trends

The web services test literature abounds with uses of model checking that were not fully anticipated by the original tool developers. This section reviews recent usage patterns of model checkers to delineate problems and trends that may influence future architectures. Figure 1 depicts how model checking is presently used in web services development. Note that model checking occurs last, after composing web services in step 3 and performing service discovery in step 2. Modeling earlier during syndication, checking later during deployment, paying increased attention to long-running transactions, and incorporating incremental model checking into a software evolution paradigm, are but a few trends that will change this picture within the next few years. We discuss each trend in what follows.

2.1. Syndication Time Modeling

The need for more targeted automatic discovery and composition of web services suggests additions to web service descriptions. The authors of [15] predict that Univer-

*Readers may contact the authors through Taghi Khoshgoftaar, Empirical Software Engineering Laboratory, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA. Phone: (561)297-3994, Fax: (561)297-2800.

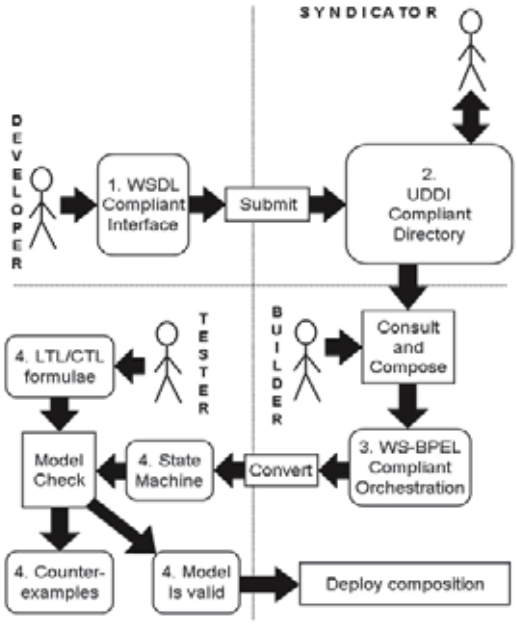


Figure 1. Present Approach

sal Description Discovery and Integration (UDDI) descriptions will be supplemented with automata-based specifications that include the model used for model checking and the Web Ontology Language for Semantics (OWL-S) based specifications containing temporal constraints.

During web services syndication and discovery, one would like to identify services that perform an "equivalent task" as a means of finding alternative web services. This implies the need to detect various forms of *bisimulation equivalence* – a functionality found in some verification environments that also happen to support model checking [3, 16].

2.2. Service Evolution

Assuring the quality of a web service composition as it continues to evolve *after* its initial deployment, requires postponing the checking phase of model checking until after Step 4 in Figure 1. The CHARMY project [4] model checks software components that evolve, but must retain *correctness by construction*. This software architecture based approach addresses already implemented software components. Incremental model checking, first described in [23] and later used in CHARMY, may lead to techniques for augmenting and visualizing the state space as one component dynamically replaces another.

The feasibility of model checking as a system evolves depends on the type of system, which includes Closed systems, Weakly-closed systems, Weakly-opened systems, and Opened systems. The authors focused on the former two, considering the latter two as less feasible.

Closed systems have a fixed set of components and a fixed connector or glue code artifact. Their web service artifacts can be "set in stone", making them the most feasible to model check. Evolution is limited to upgrading any component or its connector with those having *identical* interface and message exchange behavior. Web service artifacts remain unchanged while the implementation that operates behind the WSDL artifact (i.e. choice of middleware platform, or updated executables) changes. Its composition may be best implemented as an orchestration in WS-BPEL and model checked with existing tools at design time. Closed systems do not impact Figure 1. Although least malleable, the ability for each constituent web service to be independently upgradable showcases the strengths of SOA.

Weakly-closed systems can undergo types of reconfiguration that will require run time model checking. Such a system has a fixed set of component types or roles, yet its glue code still can dynamically bind to different yet *purportedly equivalent* services. Over time, evolution involves swapping a service instance of a given type with another instance of the same type. In web services, the peers in a long-running transaction often change but their fundamental roles, relationships, and behaviors do not, and may be best modeled as a *choreography*. For example, long-running transactions in hospital case management will always require an attending physician for any hospitalized patient. If that physician is unavailable, a different physician must temporarily assume the *role* of attending physician. The new attending physician, however, may perform rounds at a different time of day, potentially impacting related workflows. Thus in weakly-closed systems one is concerned with evolving systems using *sufficiently* equivalent services, while checking that each substitution does not adversely impact existing workflows.

Since such systems operate asynchronously, their state spaces can become intractably large, necessitating the use of incremental model checking. Although [4] proposes one of a number of strategies for dynamic composition, reconfiguration, and verification of weakly-closed systems, two facts stand out. Firstly, this dynamic incremental verification will require use of a highly optimized C Language-based model checking tool, which motivates their choice of the Spin model checker. Secondly, they observe that any dynamic reconfiguration strategy will not be appropriate for real-time systems or for systems subject to hard time constraints.

Thus, users must choose model checking approaches based on a three-way tradeoff between flexibility, assurance, and performance. The best one can hope for is to find

an approach that optimizes on these three variables, given the real world requirements for some specific web service application. In this case, service evolution can only maintain assurance at the expense of performance.

2.3. Incremental Coverage Testing

Since a model is an abstraction, it cannot be directly used as a testable artifact. However, model checkers have been used to generate test cases [8] that cover either all or at least the most common situations. Incremental coverage testing involves generating only new or deprecated test cases, to be examined on an exception basis, to determine whether the new or deprecated cases were what stakeholders had intended. The authors of [13] and [20] additionally wished to generate both positive examples and negative (counterexamples) for each temporal logic formula for each path in the state space [13], or each decision in the source code reflected in the state space [20].

Positive examples can be generated by model checking to the negation of these temporal logic properties. Not all positive examples for all coverage criteria can be generated using the Linear Temporal Logic (LTL) like that supported by Spin. Since Computational Tree Logic (CTL) does correctly handle this negation, [13] used the SMV model checker, although the UPPAAL verification environment also supports CTL [1]. Furthermore, a user needs to know if a change will require generating an intractably large number of new and/or deprecated cases, suggesting the need for a more incremental revision in the composition. If a sufficiently small increment cannot be specified, then a subset of test cases can be generated based on use cases observed in event logs, but at the expense of assurance.

Thus, use of model checkers as test case generators necessarily requires a number of workarounds. Decoupling functionalities described in Section 3 will streamline this mode of use.

2.4. Integrating Instrumentation

Instrumentation provides feedback that describes how web service compositions actually behave. Gravel, et. al. [10] consider using Spin to model check web service orchestrations, in conjunction with use of their proposed Execution Analysis tool for WS-BPEL (EA4B). This addresses three issues: (i) lack of tool support for understanding the manner in which WS-BPEL actually executes, (ii) enable exploration of service execution to identify the source of an erroneous service, and (iii) assist the user in making informed decisions on correctness of observed behavior. They propose instrumenting WS-BPEL artifacts so that post execution debugging and verification can be performed, or used for near real-time monitoring, or integrated with Spin. In the latter use case, their Web Services Analysis Tool (WSAT) [15] can translate WS-BPEL artifacts to Promela source code for input to Spin. Spin can then generate both positive and negative test cases, which can be executed by the web service composition for tracing and visualization using EA4B.

Process mining uses event logs to discover various perspectives on a process, including data flows, social interactions, and control flows. An interesting approach for generic workflow applications involves use of noise-tolerant genetic algorithms described by de Medeiros [7]. For web services, this discovery can be made easier since the block structured nature of WS-BPEL supports implementation of only a subset of workflow patterns [21]. Faithful reconstruction of a WS-BPEL artifact using process mining tests both the extent and quality of instrumentation. It provides confidence that the corpus of coverage test cases appearing in the event log used for reconstruction is complete.

2.5. Stateful Web Services

Web services must retain state as parties to a long-running transaction enter and exit, or when a web service composition must consistently maintain the state of its variables.

A different use case for model checking, involves conformance checking to some set of norms encoded in a standard like WS-BusinessActivity. Ramsokul et. al. [19] propose a test bed for transaction-oriented (i.e. stateful) web service compositions supplemented by artifacts from the WS-BusinessActivity standard. Over time, such compositions may encounter varying subsets of parties to a long-running transaction, as in case management workflows implemented as web service choreographies. Using event logs, descrepancies between it and the process model are analyzed with respect to some set of assumptions that must not change throughout the lifespan of the case. Implied in this approach is the ability to formally model and state properties in WS-BusinessActivity artifacts.

Zheng et. al. [26, 27] propose a means of automated generation of test cases from WS-BPEL artifacts that manage data dependencies. They use model checkers to automatically generate data flows for each variable by formulating test criteria as trap properties – the negations of the original properties being verified, which is a similar approach taken by [13] and [20]. When generating tests for stateful web services, [27] considers WS-BPEL *variables* and *links*, describing in detail their use of web service automata as an intermediate representation that could then be converted into either Promela or SMV.

2.6. Visualizing State Spaces

Visualization can provide insights into the structure and behavior of concurrent systems. With the availability of exchange formats described in Section 3, visualization may become more routinely used. Motivated by

this need, [11] proposes the use of the freely available mCRL2 verification toolkit that provides visualization support for massive state spaces. It can be downloaded at: http://www.mcrl2.org/wiki/index.php/Home. The specification formalism for mCRL2 is the *Pi-Calculus*, known for its ability to describe concurrent processes having configurations that may change during run time.

The mCRL2 toolkit is intended for general purpose test and verification ranging from microscale embedded systems to macroscale web services. It provides intriguing 3D visualization support for state spaces based on three topological properties: (i) proximity of each state to the initial state, (ii) the size of visual elements as proportional to size of node clusters, and (iii) retention of the symmetry of the resulting state space. On the surface, these visualizations resemble those governing the formation and organization of biological organisms, particularly dendritic phenomenon.

The toolkit includes model checking capabilities, providing state space compression for otherwise large but regularly structured concurrent systems [3]. Deadlock or other property violations are color coded, enabling developers to visualize the context of modeling or compositional errors, while test traces can display the traversal path over the state space from initial state to some designated state. Considering the trend toward decoupling, there will be a need for generating state spaces in other tools to some standard interchange format for import into the visualization component of mCRL2.

3. Architectural Features

By observing usage trends for model checking web services, we identify four features that must be incorporated into any SOA used for verifying web services. The first involves *decoupling functionalities* offered by model checkers so that users can orchestrate their own service verification environment. The second feature *emphasizes soundness* of web service compositions by representing web service artifacts so that machine verifiable models and properties can be inferred. The third requires *streamlining preprocessing*, which occupies the vast majority of time and effort. Finally, a model checking environment should *control its level of abstraction*, so that the models and properties are defined at an appropriate level of detail, subject to the size of the state space.

3.1. Decouple Functionalities

As an end-to-end process, model checking web services involves (i) converting a web service artifact to a model for model checking, (ii) converting the model to a state space, (iii) mining the state space for conformance to temporal logic properties, (iv) producing counterexamples, and (v) feeding these back through the executable web service artifact. Presently, model checkers lump together steps (ii) through (iv).

Syndication time modeling and deployment time checking requires decoupling step (iii) from (ii). For visualization to operate as its own service, it must not depend on how the state space was generated in (ii) or evaluated in (iii). Likewise, a variety of state space generation and mining techniques that optimize on size, performance, or level of abstraction have already been implemented in a number of model checkers, and should each be available as its own service. Decoupling model checking functionalities is required by the additional architectural features described in the remaining subsections.

Decoupling entails the derivation of interchange formats. Step (i) will require a SOAP message type that can represent any finite state model suitable for model checking. At minimum, this message type must be capable of expressing models used by model checkers recently appearing in the literature. These include the untimed models of Spin and SMV, the Message Sequence Charts (MSC) of the Labelled Transition System Analyser (LTSA) verification tool, and the timed automata of UPPAAL. Step (ii) will require an encoded representation of a state space that supports efficient mining of that space. The need for distribution to or access by multiple hosts for step (iii) must be considered. Step (iv) will require generating an event log containing SOAP messages suitable for the web service composition under test.

3.2. Emphasize Soundness

Sound interface definitions and behavioral specifcations at each interface will be needed in a decoupled model checking framework. Throughout the SOA verification literature, most of the models to be checked require only WSDL and WS-BPEL artifacts. WSDL is used for defining the interface of a web service between its public and private sides, while WS-BPEL is used for composing these web services into an orchestration. Due to the semi-formal semantics of WS-BPEL, a number of proposals also favored supplementing these artifacts with a formal ontology encoded in OWL-S typified by [14, 15, 25]. A similar effect may be achieved by defining a discipline for coding WS-BPEL artifacts that are correct by construction. The richness of the resulting artifacts, and hence the need for OWL-S specifications, needs further investigation.

3.3. Streamline Pre-processing

Extensive tool support will be needed for converting WS-BPEL artifacts to models suitable for checking. Currently the most prominent such tool is WSAT for converting to Spin and SMV [15]. LTSA uses an Eclipse plug-in to do this conversion into their internal representation which can thence be compared to that generated from user-specified message sequence charts [9]. Tool support for conversion for other model checkers is otherwise sparse. As

yet, no systematic study compares how well each conversion tool preserves the semantics originally encoded in the WS-BPEL artifact.

Pre-processing can avoid work later in the model checking cycle. For example, by modeling asynchronous processes as if they were synchronous can be done in identifiable cases, obviating the need to produce such large state spaces [5].

3.4. Control Level of Abstraction

What constitutes an appropriate level of abstraction has been open to debate. It becomes relevant when (i) model checking incremental changes in code and (ii) determining a suitable degree of instrumentation for process mining. If an incremental change in code caused a failure, but neither the model nor its specification changed, then either the model or the specification needs further refinement so that the code change bringing about the failure triggers counterexamples from the model checker. In process mining, a test case generated by a model checker not otherwise appearing in an event log will suggest the need for more detailed instrumentation. Likewise a test case appearing in an event log not otherwise appearing in the test cases generated by a model checker suggests the need for a more detailed model. State space size, however, constrains any such refinement.

4. A Predicted Architecture

Model-driven development of SOA's typified by the work of Heckel [12] and Lohmann [17] can be applied to developing an SOA for model checking web services over the web. A model-driven approach makes modeling complex systems accessible to practioners. In this environment, the by-product of creating a WS-BPEL artifact may be a model suitable for model checking. Model checking, in turn, can generate suites for testing the WS-BPEL artifact.

Here we sketch a model-driven approach to assuring quality of web service compositions. A web service composition has three views: (i) *artifact view* showing user-defined artifacts or graphical renderings thereof, (ii) *model view* showing its automata and property specifications, and (iii) *instrumentation view* showing an event log that, when mined, faithfully reconstructs the web service artifact. Manipulating one view will propagate changes to the remaining views. Likewise, any deleted views can be reconstructed from any remaining view.

Figure 2 shows two areas of change. The first involves step 2 in which feedback is solicited by the syndicator from the user and is further described in [24]. The second area involves the model checking process in step 4. Presently, composition is a one-off process shown in Figure 1. Figure 2 makes web service compositions reusable by requiring in step 5 the formulation of a WSDL specification for the composition, prior to interning both back into the UDDI

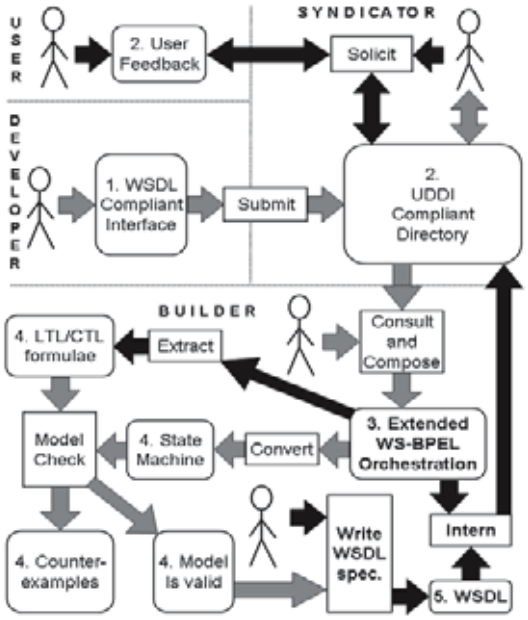


Figure 2. Proposed Approach

registry. A WS-BPEL artifact must also be coded to contain sufficient information for extraction of the remaining views. Hence step 3 of Figure 2 may require artifacts written to some instrumented extension of WS-BPEL. Note how the application builder assumes sole responsibility for verification, placing responsibility closer to the source of decision-making and aligning each role to market structures.

The result is the ability to intern valid web service compositions in addition to the descriptions of individual services that have traditionally made up the UDDI registry. Consequently, one can realize a form of hierarchical composition in which each WS-BPEL artifact can operate behind its own WSDL artifact without the knowledge of builders who may subsequently use that composition.

Someone wishing to build a web service composition works as usual with WSDL artifacts, without concern for whether any service is itself a composition. Applying this design to the Travel Agency Problem, subsidiary services of airfare and hotel can each be implemented as a non-trivial orchestration involving multiple service providers [22]. This design can compartmentalize an audition [2] or a parallel test phase to each subsidiary service composition.

The instrumentation view requires an event log that can faithfully reconstruct a WS-BPEL artifact. Serving as an alternative representation, this view must contain all coverage test cases. Whereas test suites can be generated from finite state models using model checkers, in process mining,

models may be generated from test suites.

Due to space limitations, we cannot provide more details.

5. Conclusions

This paper outlines usage trends and architectural features, suggesting a model-driven approach to verification of web services using web services. Among other things, this approach realizes a form of hierarchical composition requiring only glue code and interface artifacts. This form of composition can compartmentalize how subsidiary service compositions are deployed.

Deploying currently known state space generation and optimization techniques as individual services will enhance collaboration, by making mining and visualization of ever larger state spaces feasible. Using web service standards and ontologies to characterize the way in which state spaces are produced, encoded, represented, and mined will enable derivation of sound and workable interchange formats. This will pave the way for seamlessly integrating model checking functionalities into a web-enabled SOA.

References

[1] G. Behrmann, A. David, and K. G. Larsen. A tutorial on uppaal. In M. Bernardo and F. Corradini, editors, *SFM*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004.

[2] A. Bertolino and A. Polini. The audition framework for testing web services interoperability. In *EUROMICRO-SEAA*, pages 134–142. IEEE Computer Society, 2005.

[3] S. Blom, W. Fokkink, J. F. Groote, I. van Langevelde, B. Lisser, and J. van de Pol. μcrl: A toolset for analysing algebraic specifications. In G. Berry, H. Comon, and A. Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 250–254. Springer, 2001.

[4] A. Bucchiarone, A. Polini, P. Pelliccione, and M. Tivoli. Towards an architectural approach for the dynamic and automatic composition of software components. In *ROSATEA '06: Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis*, pages 12–21, New York, NY, USA, 2006. ACM.

[5] T. Bultan, J. Su, and X. Fu. Analyzing conversations of web services. *IEEE Internet Computing*, 10(1):18–25, 2006.

[6] E. M. Clarke, J. M. Wing, R. Alur, R. Cleaveland, D. Dill, A. Emerson, S. Garland, S. German, J. Guttag, A. Hall, T. Henzinger, G. Holzmann, C. Jones, R. Kurshan, N. Leveson, K. McMillan, J. Moore, D. Peled, A. Pnueli, J. Rushby, N. Shankar, J. Sifakis, P. Sistla, B. Steffen, P. Wolper, J. Woodcock, and P. Zave. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[7] A. K. A. de Medeiros. *Genetic Process Mining*. PhD thesis, Technische Universiteit Eindhoven, November 2006.

[8] A. Engels, L. M. G. Feijs, and S. Mauw. Test generation for intelligent networks using model checking. In E. Brinksma, editor, *TACAS*, volume 1217 of *Lecture Notes in Computer Science*, pages 384–398. Springer, 1997.

[9] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Ltsa-ws: a tool for model-based verification of web service compositions and choreography. In L. J. Osterweil, H. D. Rombach, and M. L. Soffa, editors, *ICSE*, pages 771–774. ACM, 2006.

[10] A. Gravel, X. Fu, and J. Su. An analysis tool for execution of bpel services. In *CEC/EEE*, pages 429–432. IEEE Computer Society, 2007.

[11] J. F. Groote and F. van Ham. Interactive visualization of large state spaces. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(1):77–91, 2006.

[12] R. Heckel and H. Voigt. Model-based development of executable business processes for web services. *Lectures on Concurrency and Petri Nets 2003*, 3098:559–584, April 2004.

[13] H. S. Hong, S. D. Cha, I. Lee, O. Sokolsky, and H. Ural. Data flow testing as model checking. In *ICSE*, pages 232–243. IEEE Computer Society, 2003.

[14] H. Huang, W.-T. Tsai, R. Paul, and Y. Chen. Automated model checking and testing for composite web services. In *ISORC '05: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 300–307, Washington, DC, USA, 2005. IEEE Computer Society.

[15] R. Hull and J. Su. Tools for design of composite web services. In G. Weikum, A. C. König, and S. Deßloch, editors, *SIGMOD Conference*, pages 958–961. ACM, 2004.

[16] M. Koshkina and F. van Breugel. Modelling and verifying web service orchestration by means of the concurrency workbench. *SIGSOFT Softw. Eng. Notes*, 29(5):1–10, 2004.

[17] M. Lohmann, L. Mariani, and R. Heckel. A model-driven approach to discovery, testing and monitoring of web services. In L. Baresi and E. D. Nitto, editors, *Test and Analysis of Web Services*, pages 173–204. Springer, 2007.

[18] S. Nakajima. Verification of web service flows with model-checking techniques. In *DBLP:conf/cw/2002*, pages 378–385. IEEE Computer Society, 2002.

[19] P. Ramsokul and S. Ramesh. A test bed for web services protocols. In *Second International Conference on Internet and Web Applications and Services – ICIW '07*, pages 16–22, 2007.

[20] S. Rayadurgam and M. P. E. Heimdahl. Coverage based test-case generation using model checkers. In *ECBS*, pages 83–. IEEE Computer Society, 2001.

[21] N. Russell, A. H. M. ter Hofstede, W. M. P. van der Aalst, and N. Mulyar. Workflow control-flow patterns, a revised view. Technical Report BPM Center Report BPM-06-22, Queensland University of Technology, Brisbane, QLD, Australia, January 2006.

[22] J. C. Sloan, T. M. Khoshgoftaar, and A. Folleco. Testing web services as agents. In *Proceedings of the 14th ISSAT Reliability and Quality in Design Conference*. ISSAT, 2008.

[23] O. V. Sokolsky and S. A. Smolka. Incremental model checking in the modal mu-calculus. *Proceedings of the Sixth International Conference on Computer-Aided Verification (CAV), Lecture Notes in Computer Science 818*, pages 351–363, 1994.

[24] Y. Sun, S. He, and J. Y. Leu. Syndicating web services: A qos and user-driven approach. *Decision Support Systems*, 43(1):243–255, 2007.

[25] Y. Wang, X. Bai, J. Li, and R. Huang. Ontology-based test case generation for testing web services. In *ISADS '07: Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems*, pages 43–50, Washington, DC, USA, 2007. IEEE Computer Society.

[26] Y. Zheng, J. Zhou, and P. Krause. Analysis of bpel data dependencies. In *EUROMICRO-SEAA*, pages 351–358. IEEE Computer Society, 2007.

[27] Y. Zheng, J. Zhou, and P. Krause. A model checking based test case generation framework for web services. In *ITNG*, pages 715–722. IEEE Computer Society, 2007.

A Metadata Model for Managing and Querying XML Resources in Peer-to-Peer Systems

Deise de Brum Saccol, Nina Edelweiss, Renata de Matos Galante
Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS)
Av. Bento Gonçalves, 9500, Porto Alegre, RS, Brazil
{deise, nina, galante}@inf.ufrgs.br

Abstract

Peer-to-peer (P2P) systems provide access to shared resources, such as XML files. Keyword-based P2P systems do not retrieve the necessary file if a synonym is used as part of the query. This situation happens because different but related terms may be used to describe similar information. Besides, automatic systems lack to find, extract and integrate relevant information spread over different resources. Thus, two important questions must be answered for searching and retrieving purposes: which peers have the files that satisfy the query, and how to deal with structural incompatibilities. To address these issues, our work proposes the use of annotations (represented by metadata and mapping functions) that allow defining the structural and the semantics of a document. Using such annotations provides an intelligent query processing that allows users to pose queries without being aware of the file location and structure.

1. Introduction

Data semantics defines the relations between symbols and what they denote. In particular, semantic heterogeneity refers to differences in the meaning of the data and it makes difficult to identify the relations that exist between objects in different components. The problem of semantic heterogeneity can be described as integrating structurally dissimilar but semantically equivalent objects and determining semantic equivalence of objects [1]. Thus, the semantic interoperability aims to allow the cooperation of applications that were not initially developed for this purpose.

Search engines provide automatic support for information retrieval which helps in finding data sources. However, the remaining action of extracting and using the information to solve a given task remain for the user [2]. Keyword-based systems do not retrieve the necessary file if a synonym is used as part of the query [5]. This situation happens because different terms may be used to describe similar information. Representing and managing the semantics of applications are essential for the interoperability. In order to address these issues, our approach employs annotations that allow structural and semantic definitions of documents. In this way, our proposal provides an intelligent query processing that allows users to pose queries in a P2P system without being aware of the location and structure of the files.

Moreover, another problem arises from the lack of semantics in the resources. Consider two P2P applications that need to exchange data. One possible approach is to build an adapter that transforms data and structure between them.

However, the adapter construction is a hard task that requires knowing the data organization in both applications.

Furthermore, the complexity and the developing time tend to be quadratic in relation to the number of component applications [3]. A possible solution is to employ some kind of metadata for describing the file semantics. However, this scenario states two critical questions [4]: how to deal with different concepts used to describe the same information, and how to acquire and maintain the necessary metadata to solve the vocabulary sharing issue.

The searching optimization in P2P scenarios also faces two other problems. The first problem is the existence of multiple resource representations. The existence of duplicated resources may be necessary for increasing the performance, since the user poses a query and the results are returned from a specific peer. Nonetheless, in order to take advantage of resource replication, it is necessary to manage these multiple representations.

The second problem arises from the evolutionary behavior of some resources. The evolving issue is a fundamental aspect in any persistent information system. also, it is even more evident in XML domain, with frequent structure and content changes. The evolution aspect must be managed to allow historical retrieving, for example by using versions. Although the version concept is well known for managing co-authoring on software engineering, it is still a big challenge in distributed environments.

To overcome the semantic heterogeneity issue in such systems, our work relies on the use of ontologies, metadata and mappings for interoperability enhancement. The presented approach is part of *DetVX* [6], a framework for detecting replicas and versions of XML files in P2P systems. The main contributions of this paper are:

- The specification of a metadata model for managing XML files in a P2P scenario;
- The definition of mapping transformation functions for accessing equivalent XML objects in different files.

The paper is organized as follows. Section 2 presents the motivating scenario for this work. Section 3 overviews the *DetVX* environment. Section 4 presents the metadata management. Section 5 discusses the proposed mapping mechanism based on transformation functions, using *XPath*. Section 6 details the query processor capabilities. Sections 7 and 8 discuss related work and concluding remarks, respectively.

2. Motivating Scenario

Consider a network composed of *n* peers, where peer *p1* stores the file *f1* and peer *p2* stores the file *f2*. The files are described in Figure 1 (a) and (b). Consider query *Q1*: "get the *name* of *people* that live in Garbonzoville". The goal is to guarantee that: (i) all peers that receive the requesting message are able to process *Q1;* (ii) only the peers that are able to process *Q1* receive the requesting message; (iii) the user does not worry about structural and content incompatibilities between the files; and (iv), the user does not worry about location and peer organization.

```
<resume>
    <name>
        <firstname>Hu</firstname>
        <surname>Doe</surname>
    </name>
    <address>
        <street>123 Elm #456
        </street>
        <city>Garbonzoville</city>
        <state>NX</state>
        <zip>99999-9999</zip>
        <contact>555.555.5555
        </contact>
    </address>
</resume>
```

```
<resume>
    <fullName>Jo Doe</fullName>
    <address>
        <adr>123 Elm #456,
        Garbonzoville </adr>
        <state>NX</state>
        <zipCode>99999-
        9999</zipCode>
    </address>
    <contact>
<phone>555.555.5555</phone>
<email>doe@doe.doe</email>
<url>http://doe.com/~doe/</url>
    </contact>
</resume>
```

Fig. 1. Examples of XML files, *f1* and *f2*, with structural incompatibilities

Two main problems must be solved toprocess the query [4]:

- *Resource discovery* - which files satisfy the query? Where are they located?
- *Vocabulary problem* - how to deal with structural incompatibilities? How to access the desired information?

To address these issues, two aspects are considered. First, documents with different structures can be described by the same ontology, as depicted in Figure 2.

Fig. 2. Application domain concepts described by an ontology

The user issues a query over the system by defining content and structural constraints based on the available ontologies. Then, the system is in charge of finding and retrieving the information. For instance, the query *Q1* formulated over the ontology is expressed as "get the *firstname* of *person* whose *address* contains Garbonzoville". Therefore, an ontology can be used as a common global model for describing related documents. However, in order to use an ontology to hide the heterogeneity issue in P2P systems, we need a mapping structure that captures the equivalence relations between the files and the respective ontologies.

To solve these two issues (i.e., resource discovery and vocabulary problem), our work relies on three aspects: ontologies, semantic annotations (denoted by metadata), and mapping transformations between ontologies and resources. The approach presented in this paper is part of *DetVX* [6], a framework for detecting replicas and versions of XML files in P2P systems. The *DetVX* framework is briefly described in the next section.

3. Framework

DetVX (***Det****ection of Replicas and* ***V****ersions of* ***X****ML Documents*) [6] is a framework for detecting, managing and querying replicas and versions of XML files in a P2P context. In this framework, files are stored following the super peer architecture. Files within the peers are related to a specific application domain, described by an ontology (e.g., *curriculum* domain). Peers must connect to the super peers in order to share their files. Super peers are managed by the administrative super peer. Metadata are used to represent information related to peers, super peers and files. Ontologies are used as a peer grouping criterion into super peers. [15].

The functionalities of *DetVX* are available through the following modules. The user interacts with the system through the user interface, which allows registering peers and files (using the *peer manager*). To get connected, a peer must choose a suitable peer. This is done by verifying the application domain of its files (using the *ontology manager*). After the peer connection, the framework verifies if the shared resources refer to versions or replicas of existent files already available in the network *(*using the *replica and version manager*). Finally, the user is able to submit queries (using *the query processor*).

The focus of this paper is the metadata and mapping maintenance. Such metadata and mappings are mainly generated by the peer manager and the ontology manager. Their maintenance aims to allow users to pose a query over the ontology. Then, the system:

- Verifies where the information can be found (in which file, peer and super peer), based on the ontology manager metadata;
- Translates the query to the underlying source format, based on the mapping functions generated during the file and ontology matching phase (the matching is performed by the *ontology manager*);
- Processes the query and retrieves the results;
- Translates the results to the ontology format.

A repository stores the ontology manager metadata and the mappings between the ontologies and the data sources (including some information needed to handle semantic heterogeneities), as follow.

4. Metadata Management

Our approach uses metadata to describe information about the distribution of data. Maintaining metadata is critical for query processing, since they provide valuable information that can be used for optimization. For example, they are used to manage identifiers, file registering and modification times,

file locations and some other relevant information. Metadata are represented as XML files and classified as follows.

4.1 Ontology Manager Metadata

The ontology manager metadata describes information that relates the XML files and the ontology (which describes such document). Information about ontology association is described in a XML document, as shown in Figure 3. Metadata allow specifying the ontology (attribute ontology) for a super peer (attribute id), a domain label (domain), and the file of the ontology (file). It also represents the associated peers (peers) to a specific ontology (ontology) in a certain super peer.

```
<AssociatedOntologies>
   <superPeer id="SP2" ontology="Ont1"><domain>Research
         Projects</domain><file>ResearchProjects.owl</file>
      <peers>  <peer id=”P3”><doc>F2</doc> <doc>F7</doc>
<doc>F8</doc></peer>
            <peer id=”P4”><doc>F1</doc> <doc>F5</doc></peer>
      </peers>
   </superPeer>
</AssociatedOntologies >
```

Fig. 3. Ontology Manager Metadata

These metadata are maintained whenever the *ontology and XML file matching* is performed. For a document *file* stored in a peer *id* matched to an ontology *ontology*, the metadata describe the domain (domain) to which the document belongs. Given a query belonging to a specific domain, the system accesses the metadata and identifies which super peer must get the user query. We assume that one query is related to one domain, so not domain correlation is necessary.

For instance, a query posed over the *research project* domain is formulated on the ontology *Ont1*, as described in Figure 3. This query is routed to super peer *SP2*. By looking at the metadata, the system also knows that peers *P3* and *P4* contain files that belong to the desired domain. Thus, the query is routed only to those proper peers that are able to answer it, instead of flooding the entire network.

4.2 Super Peer Metadata

Each super peer has metadata about its aggregate peers and the respective files. These metadata are structured as an XML file, as shown in Figure 4.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE Metadata SYSTEM "C:\METADATA.dtd">
3  <Metadata superPeerId="SP1">
4    <document docID=”D1” fileID="F7" HDoc=”YES”>
5       <version versionID="1" peerID="P1" registeringTime="10/10/2005"
6          modificationTime="08/08/2004" duplicate="no"
7          hashResult="d49622ddab3733549e54749755fd52b5">
8          <element name="author" TS="08/08/2004" TE="10/15/2004"/>
9          <element name="address" TS="08/08/2004" TE="10/15/2004"/>
10
11      </version>
12      <version versionID="2" peerID="P2" registeringTime="11/20/2005"
13         modificationTime="10/16/2004" duplicate="yes"
14         hashResult="7c00bb062edc60fa548729a3d55c04fd">
15         <locationDuplicate>Peer 3</locationDuplicate>
16         <element name="author" TS="10/16/2004" TE="now"/>
17         <element name="address" TS="10/16/2004" TE="now"/>
18         <element name="phone" TS="10/16/2004" TE="now"/>
19      </version>
20   </document>
21   <document fileID="D8">…</document>
22 </Metadata>
```

Fig. 4. Super peer metadata

Those metadata define the available versions and replicas of XML files in a specific super peer (superPeerId), and the corresponding timestamps for each element (TS, TE) that is found in a certain document instance (fileID) in a peer (peerID). The metadata information is updated whenever a new file is registered and is used during the querying process.

We assume that each element has two timestamps, *TS* and *TE*, inferred from the file modification time in which the element is represented.

4.3 Administrative Super Peer Metadata

The administrative super peer metadata are presented in Figure 5. They describe some file information, such as identifiers, location and registering/modification time. Specifically, this metadata represent the existent super peers (ID) and its current aggregate peers (peerID), the local identifier (fileID), its hash result (hashResult), registering time (registeringTime), and modification time (modificationTime) for each file in each peer. The registering time denotes the time that the file was shared within the system. The *fileID* attribute is a value mapped from the hash result. The document identifier is denoted by *docID*.

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
   <superPeer ID="SP1"><message peerID="P1">
         <document docID=”D1” fileID="F1"
         hashResult="ece50ed4d6d48dac839bfe8fa719fcff"
         registeringTime="10/10/2005"
         modificationTime=08/08/2004”/>
         <document docID=”D2” fileID="F2"
         hashResult="e3732b09b5b2a9aa452b8ef7802db638"
         registeringTime="10/15/2005"
         modificationTime=12/12/2004”/>
         <document docID=”D3” fileID="F3"
         hashResult="73cbe8e94c7fa839ba1246b34b2a49cd"
         registeringTime="10/20/2005"
         modificationTime=10/08/2005”/></message>…
   </superPeer>
   <superPeer ID="SP2">....</superPeer>
</metadata>
```

Fig. 5. Administrative super peer metadata

The metadata are accessed in the following situations:

- The system needs to verify peer modifications - peer changes are detected when the hash function result returns a different value for a specific file, compared to the value stored in the metadata. We also consider the file modification time to optimize this process, by calculating the hash result only if the modification time has changed.
- The system needs to update the metadata at the first time peer connection – in this case, the metadata are updated with information related to the connection, such as: peer and super peer identification and hash results, registering time and last modification time of each file.
- The system needs to update the metadata after a peer reconnection – this is the case when a peer changes some files and reconnects to the system.

5. Mapping Management

Instances corresponding to terms in the ontology are stored in the underlying XML documents. Therefore, the mapping information relates terms in the ontology to data elements of the underlying XML files. Mappings are the information needed to retrieve the underlying data corresponding to each

term in the ontology. The management involves generating and storing information about those mappings.

Two general approaches can be used for specifying links among ontologies and data repositories and for using them on query reformulation [7]: global-as-view (GAV) and local-as-view (LAV). This proposal assumes the former approach. In the GAV approach [10], for each term in the semantic view, a query over the data repositories is written specifying how to obtain instances from the data repositories. Thus, each term in an ontology is associated to the mapping information that relates that term to the underlying XML elements. The mappings are defined/updated whenever the *ontology and document matching* task is performed and they are important for query transformation, as shown in Section 6.

For each term in the ontology the mapping information is represented as a transforming function. Mapping definition for a term in the ontology is a list of mappings (one mapping for each underlying document with different structure). The transforming functions are represented in *XPath* [8]. Besides relating concepts, the transforming functions also can change values of an element or attribute such that the meaning of the information is unchanged. If underlying data are stored in one format and we have defined a concept in a different format, the mappings should specify a function that transforms such representations.

For example, consider the ontology presented in Figure 2 and the XML files shown in Figure 1. Some mappings are described in Figure 6. Assume that the element address in the ontology corresponds to information related to street, number, city and state. Where: concat function returns the concatenation of its arguments; substring-after returns the substring of the first argument string that follows the first occurrence of the second argument string. The power of the mapping expressiveness is limited by the *XPath* expressiveness. However, *XPath* allows several interesting mappings, such as: relative and absolute paths; predicates (i.e., filters); functions for manipulating strings, numbers, and booleans; node set functions (e.g., last and first position), etc.

Ontology concept	*XPath* mapping	
	Document 1 (a)	Document 1 (b)
Person		
Person/resume	//resume	//resume
Person/Name	/resume/name	/resume/fullName
Person/Name/ Firstname	/resume/name/firstname	substring-before (/resume/fullName, " ")
Person/Name/ MiddleName	-	-
Person/Name/ LastName	/resume/name/surname	substring-after (/resume/fullName, " ")
Person/Name/ Suffix	-	-
Person/Address	concat(/resume/address /street,../city,../state)	concat(/resume/addre ss/adr, ../state)

Fig. 6. Transforming functions in XPath for the resumé ontology

The mappings are generated by the ontology manager (Section 3) during the *document and ontology matching* activity. The matching task aims to establish correspondences among the ontologies and the XML documents. This task determines the overlapping concepts, and the concepts that are similar in meaning but have different name or structure.

It is possible for an ontology term to have alternative mappings. For example, when the extension of such term is stored in several XML documents. In general, alternative mappings can be defined when there are different ways to access the data corresponding to a term in an ontology. Also, alternative mappings can be used when some document is not available or reachable (e.g., if the peer is off-line) and due to optimization issues. In this situation, the creator of the ontology can establish a priority for each alternative mapping based on the access time or computation time [4], which depends on the complexity of the mapping itself. When the system cannot access the repositories corresponding to the mapping of a term, it tries to use alternative mappings. Also, only when all the possibilities are exhausted is the term classified temporally as a term without mapping.

We assume that the mappings are static and do not change over time. However, when there is a change in the structure of an underlying document, an addition of a new document with different structure, or a deletion of an existent document, the mappings to the associated ontology should be changed. In our case, all we need is to modify the mappings of the ontology that describes that document. If a change in a repository alters its semantics, only the obsolete semantic relations must be updated.

6. Query Processor

The query processor asks the ontology manager for a graphical representation of a given ontology, to allow the user to navigate and select one to edit the query. Before forwarding the edited query to a specific peer, the system should know which files each peer has. If the posed query requires the current status of an element or attribute, only peers that have the last version of that document should receive the request. Therefore, the whole system is not flooded with requests that only specific peers are able to respond. To evaluate which files must be queried to answer a specific request, our approach proposes the use of metadata, presented in Section 4. Such metadata are accessed to optimize the searching process, as follow described.

6.1 Querying the Metadata

The infrastructure available in the proposed environment is enough to allow metadata analysis and query routing. The defined metadata are used for this purpose. For example, consider the queries *get the first version of the person address* and *get the last version of the person address*. The query submission works as follows: the user poses a query in a specific peer (named *querying peer*). This query belongs to a specific domain, such as *resumé* domain. There are two situations that must be considered:

1. The query domain and the peer domain are the same, represented by D. In this case, if the querying peer is able to answer, the query is computed and the results are returned. Otherwise, the query is routed to the super peer, which is responsible for defining the routing, based on the available metadata.
2. The query domain and the peer domain are *not* the same. In this case, the query must be routed to a suitable super peer that aggregates documents belonging to the domain D.

Based on super peer metadata, as described in Figure 4, it is possible to access the version v_i of an element e_j, the history of an element e_j, the version v_i of a document D_i, the history of an element e_j between the time interval x and y, and some other temporal queries. For example:

1. Retrieve the version v_i of an element e_j – for instance, get the first version (versionID="1", line 5) of the element *author* (element name="author", line 8). By searching the version number represented in metadata, the system can verify that the first version of the queried element is found in *peer* 1 (peerID="P1", line 5) located at *super peer* 1 (superPeerId="SP1", line 3). Thus, the system must access this document and returns the results.
2. Retrieve the version v_i of a document D – for instance, get the second version of the document *D1*. To answer this query, the system searches the desired version (versionID="2", line 12) of the document (docId="D1", line 4). Looking at the metadata, the system verifies where this version is located and accesses the specific file.

6.2 Querying the XML Files

The query processor evalutes queries formulated over an ontology by translating the query and accessing the necessary underlying data. The following steps are executed to obtain the data corresponding to a user query:

- *Query construction* - the user formulates a query using the terms of the ontology (*Qm*);
- *Query translation* - the query formulated over an ontology is rewritten (*Qm*) in terms of the XML files;
- *Data retrieval* - the mapping expression is sent to the proper peer, data are accessed, individual results (*Ru*) are combined, and the final answer (*Ro*) is returned to the query processor. This step is represented by the activities *data access* and *result translation*.

This process is depicted in Figure 7.

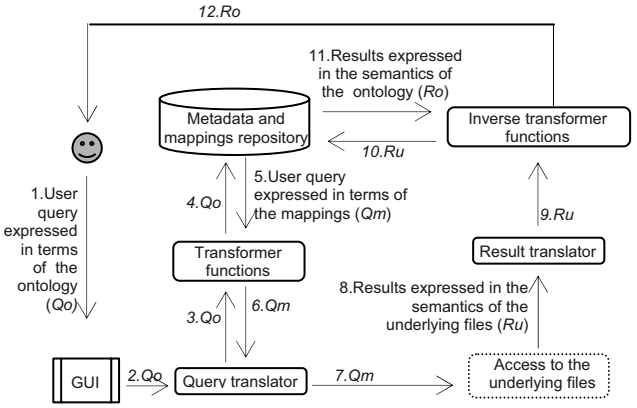


Fig. 7. Query processing flow

The tasks on query processing are as`follows:

- Query construction - the user builds a query through a user interface. For that, the system shows the available ontologies in order to restrict the query domain. After choosing a specific ontology, the user can navigate it and build the query. No extra user intervention is necessary. The query is represented in *XPath*.

For instance, the query *Qo* "*get the name of people that live in Garbonzoville"* is represented as:

/Person/Name/Firstname [contains(../../address, "*Garbonzoville*"]

Where: function *contains* returns true if the first argument string contains the second argument string.

We assume that users formulate queries over only one ontology. After obtaining the user query, the query processor invokes the query translator. The query translator uses the pre-defined mapping information that relates terms in the ontology to terms in the XML files. The data are later retrieved, with the help of the transformer functions and the metadata/ mapping repository.

- Query translation: the system takes a query against the ontology, transforms it into several mapping expressions, and executes the queries in the underlying files. When submitting a query on a P2P system, the user should consider only the data semantics, without worrying about aspects related to syntax, location, structure, and data repositories; those issues are handled by the system.

For instance, using the mapping definitions presented in Figure 6, the query *Qo* is translated to two expressions:

Qm1:

/resume/name/firstname[../../address/city= "*Garbonzoville*"]

Qm2:

substring-before (/resume/address/adr
[contains(.,"Garbonzoville")]/../../fullName," ")

- Data access: the system retrieves the data that correspond to the query and that are valid according to the chosen ontology. For this task, the system uses the mapping information to translate the user query into different sub-queries for the underlying XML documents. By looking at the metadata presented in Section 5, the system identifies which XML documents are related to the chosen ontology, on which peer they reside, and how to access them.

Data are retrieved from the different repositories in different structures and data formats. Therefore, different sub-answers should be combined (correlated) and presented to the user. The correlation step has two goals [4]: to solve the heterogeneity at structural level, and to join answers coming from different files. Repository answers still follow the structure of the specific XML source. Thus, each repository answer is changed from the structure of the file into the structure of the corresponding ontology supported by such a file. Then, after this process of applying inverse transformer functions, all the repository answers are expressed in the format of the ontology that describes such repositories. So, after correlating the different repository answers, the result is returned to the query processor.

In order to correlate sub-answers, we have to deal with two issues: (i) *entity identification* - how one identifies representations of the same real-world entity in different files; and (ii), *attribute-value conflicts* - how one deals with differences in data values among attributes that represent the same real-world entity? The previous work presented in [9] discusses the proposed approach for solving these issues.

- Result translation – the transformer functions are also defined to solve the vocabulary problem at the structural level, For instance, the query *Qm* produces the following results:

Ru1: <firstname>Hu</firstname> *Ru2:* Jo

By considering the mapping expressions in Figure 6, the following transformation is performed:

Ru1(<firstname>Hu</firstname>*)*→*Ro1:* Firstname: Hu

Since the second result is only a substring (*Ru2:* Jo), the returned answer for this file is *Ro2:* Jo.

7. Related Work

Several research works have addressed the problems of how to specify the links (i.e., the mappings) among semantic views

and data repositories, and how to use them for query reformulation. In the GAV approach [10], for each term in the semantic view, a query over the data repositories is written specifying how to obtain instances from the data repositories. The LAV approach [10] takes the opposite way: for every data repository *D*, a query over the terms in the semantic view is written for describing the instances found in *D*. The main advantage of the GAV approach is that query reformulation is simple, since it reduces to view unfolding.

In [11], a data model is proposed for encoding semantic information that combines features of ontology with a description and rating model that allows handling heterogeneous views on the domain of interest. [12] describes the *AutoMed* repository and some associated tools, which provide the implementation of the both-as-view (BAV) approach to data integration. They also describe transformations between those data modeling languages. [13] proposes *TIQS*, an approach to data integration that uses semi-automatic schema matching to produce source-to-target mappings based on a predefined conceptual target schema.

Although several data integration systems have been proposed to address these problems, no single system addresses all the important issues (such as heterogeneity, scalability, change of local information sources, query processing complexity, and global schema evolution) in a unified approach. Our approach handles heterogeneity by using mapping functions and it is designed for preserving the scalability (since it has been proposed for a p2p scenario). Furthermore, changes of local sources are a constant requirement, since peers get on-line and off-line constantly and the shared files can also change. Finally, the global schema evolution is easily achieved by updating the ontology and the necessary mappings.

8. Final Remarks

Technology such as search engines in the WWW currently supports automatic information retrieval that helps in finding information sources. However, the remaining tasks of extracting the information and using the information to solve a given task remain for the human user. There are severe bottlenecks that must be passed in order to overcome the current lack of standards in the internet, such as: lack of ways to represent and translate and lack of a means for content descriptions. In consequence, there is a clear need and a large commercial potential for new standards for data exchange and domain modeling [2].

This paper presented two main contributions: the specification of a metadata model for managing resources in a P2P scenario; and the definition of mapping transformation functions to allow accessing equivalent resources in different resources. The key point of our proposal is the use of ontologies. Ontologies are vital for developing the semantic web.

We are currently developing a graphic tool for peer management based on JXTA platform [14]. The system will allow managing the super peers, peers and corresponding shared files. We are also working on the prototype that implements the metadata and the mappings presented in this paper. The conclusion of the prototype implementation will allow assessing the system performance and scalability.

Acknowledgments

This work has been partially supported by CNPq under grant No. 142396/2004-4 for Deise de Brum Saccol. It is also supported by Pronex FAPERGS under grant No. 0408933 and CNPq under grant No. 481055/2007-0 for Renata de Matos Galante. We would like to thank Mirella Moura Moro and Eduardo Kessler Piveta for their helpful comments.

References

1. Elmagarmi, A., Rusinkiewicz, M., and Sheth, A (ed). Management of Heterogeneous and Autonomous Database Systems. San Francisco: Morgan Kaufmann, 1999
2. Fensel, D. Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce. Springer, 2001
3. Staab. S. and Studer, R. Handbook on Ontologies (Intl. Handbooks on Information Systems). Springer, 2004
4. Mena, E., and Illarramendi, A. Ontology-based query processing for global information systems. Springer, 2001
5. Freenet. Available at http://freenet.sourceforge.net
6. Saccol, D.B., Edelweiss, N. and Galante, R.M.. Detecting, Managing and Querying Replicas and Versions in a Peer-to-Peer Environment. In: 1st IEEE TCSC Doctoral Symposium (7th IEEE Intl. Symp. on Cluster Computing and the Grid), Rio de Janeiro, 2007
7. Florescu 98 : D. Florescu, A. Levy, and A.Mendelson. Database techniques for the world wide web : a survey. In SIGMOD Record, 1998
8. XML Path Language (XPath) 2.0. W3C Recommendation 23 January 2007. Available at http://www.w3.org/TR/xpath20/
9. Saccol, D., and Heuser, C. Integration of XML Data. Lecture Notes In Computer Science, V. 2590. In: Proc. of the VLDB - Workshop on Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web, 2002
10. Lenzerini, M. Data integration: A theoretical perspective. In: Proc. of the Symposium on Principles of Database Systems, ACM, 2002
11. Broekstra, J., Ehrig, M., Haase, P., and et al. A Metadata Model for Semantics-Based Peer-to-Peer Systems. In: Proc. of the WWW'03 Workshop on Semantics in Peer-to-Peer and Grid Computing, 2003
12. Boyd, M.., Kittivoravitkul, S., Lazanitis, C., McBrien, P., and Rizopoulos, N. AutoMed: A BAV Data Integration System for Heterogeneous Data Sources. In Proc. of the 16th Advanced Information Systems Engineering Conference, Latvia, 2004
13. Xu, L. and Embley, D. Using schema mapping to facilitate data integration. 2003
14. Gong, L.. JXTA:A Network Programming Environment. IEEE Internet Computing, 5(3):88–95, May 2001.
15. Saccol, D.B.. et al. An Ontology-based Approach for Semantic Interoperability in P2P Systems. Accepted for presentation in: ICEIS - International Conference on Enterprise Information Systems, Barcelona, Espanha, 2008.

Minimal Observability for Transactional Hierarchical Services *

Debmalya Biswas and Blaise Genest
IRISA/INRIA & CNRS, Campus de Beaulieu, 35042 Rennes Cedex, France
{firstname.lastname}@irisa.fr

Abstract

For complex services, logging is an integral part of many middleware aspects, especially, transactions and monitoring. In the event of a failure, the log allows us to deduce the cause of failure (diagnosis), recover by compensating the logged actions (atomicity), etc. However, for heterogeneous services, logging all the actions is often impracticable due to privacy/security constraints. Also, logging is expensive in terms of both time and space. Thus, we are interested in determining the absolute minimal number of actions that needs to be logged, to know with certainty the actual sequence of executed actions from any given partial log. This problem happens to be NP-Complete. We consider complex services represented as a hierarchy of services, and propose a decomposition mechanism which dramatically decreases the complexity (up to 2 exponentials).

1. Introduction

An interesting problem for complex systems is to determine a minimal set of actions that needs to be observable such that a given property holds. Some of the properties studied in literature of discrete event systems are normality [6], observability [5], state observability [9], diagnosability [13], etc. Our system corresponds to a composite (workflow) Web service. A Web service [1] refers to an online service accessible via Internet standard protocols. A composite service, composed of already existing (component) services, combines the capabilities of its components to provide a new service. A service schema which specifies the execution order of its components, can be modeled as a graph, performing actions on global variables. We do not tackle here the modelization of a service as a graph, which should be handled with care to yield a graph of reasonable size (see [15] and example 1).

Our long-term objective is to provide a transactional framework for (composite) Web services. A transaction can be considered as a group of actions encapsulated by the operations Begin and Commit/Abort, having the following properties: Atomicity (A), Consistency (C), Isolation (I) and Durability (D). Here, we focus on the atomicity aspect, that is, either all the actions of a transaction are executed or none. In the event of a failure, atomicity is preserved by compensation [3, 14]. Compensation consists of executing the compensating actions, corresponding to each executed action of the failed process, in reverse order of the original execution. Many advanced transactional models have also been proposed, e.g. "semantic compensation" [14] which do not require any knowledge of the execution sequence. However, their application to more autonomous systems like Web Services has been limited, where the default compensation mechanism of the widely used Business Process Execution Language (BPEL) is to "execute the completed actions in reverse order". Thus, for compensation to be feasible, we need to reconstruct each executed action or the complete history of any execution. To achieve that, we maintain a log of the observable actions. In addition to the obvious space overhead of logging, the complete log may not always be accessible. For a composite service, the providers of its component services are different. As such, their privacy/security constraints may prevent them from exposing (part of) the logs corresponding to the execution at their sites. Also, heterogeneity may lead to the logs being maintained in different formats, rendering some of them incomprehensible. Existing Web services specifications to provide transactional guarantees, such as WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity [16], are basically distributed agreement protocols which are based on the assumption that "all state transitions are reliably recorded" and can be compensated. Our approach is targeted towards a more heterogeneous environment where all transitions may not be observable. Hence, we want from a partial log of the observable actions to know with certainty the actual sequence of executed actions, to be able to compensate it.

Section 2 introduces the required formal preliminaries including the precise problem statement. Clearly, we are interested in logging the smallest number of actions possi-

*This work is supported by la Region Bretagne (CREATE ACTIVE-DOC) and ANR-06-MDCA-005 DOCFLOW.

ble. However, determining the minimal number of actions to log, such that any execution of a system is compensable, is NP-Complete. This is not very surprising, since the closely related sensor selection problems [17, 8] are also NP-Complete (see section 3). Also, the problem cannot be approximated [11] in polynomial time, which means that polynomial time algorithms cannot give a minimal set for all graphs, and that for many graphs, the observable set produced would be much bigger than the minimal set.

A complex service is often constructed hierarchically (see section 4), with some services at a high level corresponding to many composite services at a lower level. Each hierarchical level potentially describes the interactions at a different level of abstraction, e.g., the top level may describe the interactions between several providers, then the next level between services of a provider, and so on. Moreover, components can be reused in a hierarchical system, giving a compressed way to represent big systems. Hierarchical systems are often used for words [10], Finite State Machines [2], and even trees [7]. For words, e.g., hierarchical structures correspond to the LZ compression [10]. We show in section 5 how to use the hierarchical representation to compute efficiently a minimal observable set of transitions. The algorithm is not straightforward since the log of both minimal sets of actions of different components is not necessarily enough to recover the actual sequence of executed actions of the whole graph. One solution could be to resort to function summarization, but then only an overapproximation of the minimal set of actions would be obtained. Nevertheless, *we show that it suffices to run the algorithm with slightly different parameters on each component*. We thus obtain a divide and conquer algorithm. We present a theoretical complexity analysis which illustrates the benefit of our method (up to two exponentials better when using the full hierarchical representation and one exponential better by using the hierarchical representation even if components are used only once, compared to flattening the hierarchical graph), that is verified experimentally (section 6). Proofs and details omitted for lack of space can be found in [4].

2. Preliminaries

Formally, we model a transactional service as a 4-tuple $M = (Q, s_0, s_f, \mathcal{T})$, where (Q, T) is a graph ($q \in Q$ is called a state and $t \in \mathcal{T}$ a transition) and $s_0 \in Q$ and $s_f \in Q$ are the initial and final states, respectively.

Our systems are thus graphs with a unique input and output point, each node and arc corresponds to a state and transition, but we ignore the alphabet. We assume that the service M does not have any unreachable states and that all states can reach the final state s_f. For convenience, we also assume that there are no outgoing edges from s_f and no incoming edges to s_0.[1] We say that an execution sequence $\rho = \tau_1 \cdots \tau_n \in \mathcal{T}^*$ is a path of M if there exists $q_0, \cdots, q_n \in Q^{n+1}$ with $\tau_i = (q_{i-1}, q_i)$ for all $1 \leq i \leq n$. A path is called initial if furthermore $q_0 = s_0$. We denote by $\mathcal{P}(M)$ the set of initial paths in M. Finally, we denote by $|M|$ the size of M, that is, its number of transitions.

In general, for any execution ρ, we call observation projections the observation we have after ρ was executed (a sequence of actions, control points, data ...). We say that an observable projection σ is uncertain if there exists two paths having the same projection. The service M is execution sequence detectable iff none of its observable projections are uncertain.

Definition 1 *Let $\mathcal{T}_O \subseteq \mathcal{T}$ be the set of observable transitions. The observation projection $Obs_O : \mathcal{T}^* \longrightarrow \mathcal{T}_O^*$ is the morphism with $Obs_O(a_1 \dots a_n) = o_1 \cdots o_n$ with $o_i = a_i$ if $a_i \in \mathcal{T}_O$, and $o_i = \epsilon$ if $a_i \in \mathcal{T} \setminus \mathcal{T}_O$, with ϵ the empty word.*

That is, $\text{Obs}_O(\rho)$ is the subsequence of ρ obtained by eliminating from ρ every occurrence of a tuple which is not in $\mathcal{T}_O$. With such an observation projection Obs_O, the only way of having execution sequence detectability is to have every transition observable. Indeed, as soon as there exists even one non-observable transition, the service is not execution sequence detectable. Else, let us take a path $\rho\tau$ with the last transition $\tau \notin \mathcal{T}_O$. Then, $\text{Obs}_O(\rho\tau) = \text{Obs}_O(\rho)$. An usual way to overcome such a problem is to ask for certainty only up to the last few events of the sequence [9]. However, this workaround does not make sense in our framework since if we cannot compensate the very last action, then we cannot compensate any action at all. As such, we design a new observation mechanism, where the last control point reached before failure is monitored, even if the last action is not logged. In practice, it means that every state that is reached is monitored, and overwrite the previous state in a special memory buffer.

Definition 2 *Let M be a service, $\mathcal{T}_O \subseteq \mathcal{T}$. The observation projection $Obs_O^{last} : \mathcal{T}^* \longrightarrow (\mathcal{T}_O^*, Q)$ is the function $Obs_O^{last}(\rho) = (Obs_O(\rho), q)$ for all $\rho \in \mathcal{P}(M)$ ending in q.*

We will stick with this definition of observability for the rest of the paper. As mentioned before, we are interested in logging as few transitions as possible.

Problem statement. Given an service $M = (Q, s_0, s_f, \mathcal{T})$, we call $\mathcal{T}_O$ an observable set of transitions if the service is execution sequence detectable with $\text{Obs}_O^{\text{last}}$. We want to determine a minimal observable set of transitions $\mathcal{T}_O \subseteq \mathcal{T}$.

The cardinality of such a minimal observable set $\mathcal{T}_O$ of a service M is referred to as its observable size $MO(M) =$

[1] Notice that we could deal with a service without these requirements, but the proof would be more technical.

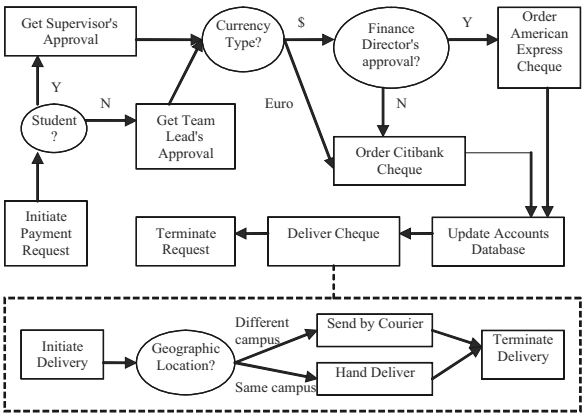


Figure 1. Travel funds request workflow.

$|\mathcal{T}_O|$. Notice that as is usual with decision and computation algorithms, it is sufficient to have an algorithm which from a service gives its observable size. That is, we can derive a minimal observable set of the service based on knowledge of its observable size in polynomial time.

Example 1 *We consider in Fig. 1 a travel funds request service, inspired by the workflow in [12]. It involves different departments across organizations, and it is hierarchical in that the deliver cheque service is hierarchically described.*

We model the service using the service $M = (S, s_0, s_f, \mathcal{T})$*, as shown in Fig. 2. Notice that this service is a simplification, since for instance the choice between the team leader or supervisor approvals is not represented. The reason is that they are both associated with an empty compensating transition, hence knowing which path was taken here is not necessary to be able to perform recovery. However, it is necessary to know which bank issued the cheque in order to be able to compensate it, by a "Cancel Last American Express (Citibank) Cheque". Note that we do not exclude the logging of data values (in some persistent storage) required for compensation. For instance, if there wasn't any "Cancel Last Cheque" mechanism, then it would be needed to log the amount and account number associated with the ''Update Accounts Database" transition. Recovery would manually credit the amount of money written in the log to the corresponding account. Obviously, we cannot save on logging the data values, but we optimize the logging associated with the path visited. Our experiments performed on BPEL representations of some workflows reveal that one transition out of five is logged (which is confirmed in section 6) and that data values logs are small compared to logging the path.*

Now, let $\mathcal{T}_O = \{e_2, e_3, e_9\}$ *and a failure occurs while processing* e_8*, that is, the cheque is not issued or delivered correctly. Then,* $Obs_O^{last}(e_1e_2e_5e_7) = (e_2, s_5) = Obs_O^{last}(e_1e_2e_4e_6e_7)$*. Thus, we do not know if an American Express or Citibank cheque was processed. With* $\mathcal{T}'_O = \{e_2, e_6, e_9\}$*, we have* $Obs_O^{last}(e_1e_2e_5e_7) = (e_2, s_5) \neq Obs_O^{last}(e_1e_2e_4e_6e_7) = (e_2e_6, s_5) \neq Obs_O^{last}(e_1e_2e_4e_6) = (e_2e_6, s_4)$*, and* $\mathcal{T}'_O$ *is an observable set of transitions. Every path from* s_0 *to* s_f *uses at least one transition from* $\mathcal{T}'_O$*.*

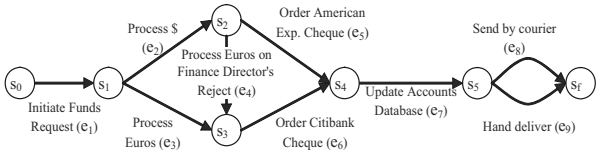


Figure 2. Modelization of Fig. 1.

3. Problem Hardness

We first relate the problem of computing $MO(M)$ using our definition of observable projections with other known problems. We state now that computing the minimal observable set is equivalent to the *uniconnected subgraph problem*, also called the *minimal marker placement problem* [8], in the meaning of the following proposition.

Proposition 1 *Let* M *be a service and* $\mathcal{T}_O$ *a subset of transitions of* M*. Denote by* M' *the service* M *obtained by deleting all transitions belonging to* $\mathcal{T}_O$*. Then,* $\mathcal{T}_O$ *is an observable set of* M *iff there does not exist a pair of paths* $\rho_1 \neq \rho_2$ *of* M' *with* ρ_1 *beginning and ending at the same states as* ρ_2*.*

To prove proposition 1, it suffices to prove that if there does not exist a pair of paths $\rho_1 \neq \rho_2$ of M' with ρ_1 beginning and ending at the same states as ρ_2, then from any observable projection (σ, q_{n+1}), we can reconstruct in a unique way a path with $\text{Obs}_O^{\text{last}}(\rho) = (\sigma, q_{n+1})$. The converse is trivial. Indeed, it suffices to define the only path ρ_i of M' between q'_i and q_{i+1} for $\sigma = (q_i, q'_i)_{i=1}^n$, and $i = 0 \cdots n$ (we fix $q'_0 = s_0$ the initial state of M', and recall that q_{n+1} is the last observed state). Then, the path $\rho = \rho_0(q_1, q'_1)\rho_1 \cdots (q_n, q'_n)\rho_n$ is the only path with $\pi_O^{\text{last}}(\rho) = (\sigma, q_{n+1})$. The search for each path ρ_i can be performed in linear time by a simple depth first search of M'.

The fact is that the marker placement problem is an NP-Complete problem. The question is then to know if there is a structural subclass of graphs which has a tractable algorithm to give the minimal observable size. We know from [8] that the minimal marker placement problem is NP-Complete even for acyclic graphs. However, the proof uses a graph with unbounded (in and out) degree. We show that the problem is NP-Complete even if the graph is both acyclic and the sum of its in and outdegree bounded by 3

(that is, indegree 2 and outdegree 1, or vice versa). The core of the proof follows the same strategy as [8], but the encoding to get a unique starting and ending point is both easier to understand and allows a lower in and outdegree.

Theorem 1 *Let M be a service, and k a number. Knowing whether $MO(M) \leq k$ is NP-Complete, even if the corresponding graph is acyclic and the sum of in and outdegree of every node bounded by 3.*

This theorem does not mean that the problem is impossible to solve, but that it cannot be solved for all possible services. For instance, the complexity of the brute force method which generates every subset of transitions and tests whether it is observable, is $O(2^{|M|})$ for a service M with $|M|$ transitions. The question then is which structural property makes the problem easier to solve and often holds for (real life) composite services. We propose hierarchical services as a candidate property.

4. Hierarchical Services

Hierarchical services provide an efficient way to model large and complex services by allowing a modular decomposition. We consider hierarchical services where two transitions (supertransitions) can be further refined into another service. A hierarchical service H is a finite sequence $\langle M_i \rangle_{i=1\cdots n}$, where $M^i = (Q^i, s_0^i, s_f^i, \mathcal{T}^i, (\tau_1^i, k_1^i), (\tau_2^i, k_2^i))$ is defined as follows:

- $(Q^i, \mathcal{T}^i)$ is a finite graph,
- s_0^i and s_f^i are the initial and final states, respectively,
- $\tau_1^i, \tau_2^i \in \mathcal{T}^i \cup \{\epsilon\}$ are two supertransitions representing services $M^{k_1^i}, M^{k_2^i}$ respectively, with $k_1^i, k_2^i > i$.

For instance, the workflow in Fig. 1 can be described by a hierarchical service $\langle M_1, M_2 \rangle$, where M_2 is made of an initial and final state, and two transitions e_8, e_9 from the initial to the final state. The service M_1 is very similar to Fig. 2, except that there is a unique transition e_{10} between s_5 and s_f instead of two. This is a supertransition (τ_1^1, k_1^1), with $\tau_1^1 = e_{10}$ and $k_1^1 = 2$, meaning that e_{10} represents M_2.

With each hierarchical service H, we associate an ordinary service $\mathcal{H}$ obtained by taking M^i, and recursively substituting each supertransition by the service it represents. For example 1, $\mathcal{H}$ is depicted in Fig. 2. Given a hierarchical service $\langle H_n \rangle$, $\mathcal{H}_j$ is a component of $\mathcal{H}_i$, if there is a supertransition (t, j) in H_i. We define the size $|H|$ of a hierarchical service H as the sum of the number of transitions of its components M^i. Its diameter $||H||$ is the number of transitions of $\mathcal{H}$. The diameter $||H||$ of H can be exponential in the size of H, because components can be reused several times (for instance, a supertransition of H_3 and two supertransitions of H_4 can represent H_{10}, in which case one does not need to redefine H_{10} three times).

Now, let us define a hierarchical system H with two levels. The top level H_1 has two states, one initial and one final, with two transitions τ_1, τ_2 from the initial to the final state. Transition τ_2 is a supertransition. It is not easy to determine a minimal set of transitions for H. Consider first that τ_2 describes a system H_2 similar to H_1, that is two transitions τ_3, τ_4 from the initial to the final state, but without supertransitions. The set $T_2 = \{\tau_3\}$ is a minimal observable set of transitions for H_2. Now, looking at H_1 as a normal system (without supertransitions), $T_1 = \{\tau_1\}$ is also a minimal observable set of transitions for H_1. We have furthermore that $T_1 \cup T_2$ is a minimal observable set of transitions for H.

However, if we take H_2' to be the system described in Fig. 2 and the associated minimal observable set $T_2' = \mathcal{T}_O' = \{e_2, e_6, e_9\}$ of transitions described in example 1, then $T_1 \cup T_2'$ is not minimal among the observable set of transitions for H. The reason is that T_2' is already an observable set of transitions, because all paths that pass through H_2 use at least one transition in $\mathcal{T}_O'$, so they can be differentiated from the path τ_1. That is, the fact that a subset of transitions is a minimal observable set of transitions is global to the whole graph, not local.

5. Algorithm for Minimal Observability

We turn now to defining an algorithm which uses the hierarchical structure of a complex service to compute the minimal observable set. First, we need the following notations. Given $\mathcal{T}_O$, a path ρ is said to be an unobserved path if it does not use any transition of $\mathcal{T}_O$. For a service M and a set of transitions $\mathcal{T}_O$ of M, we define the following predicates:

- $P_0(M, \mathcal{T}_O)$ holds if there does not exist more than one unobserved path between any two states $s_1 \neq s_2 \in Q$ ($\mathcal{T}_O$ is an observable set of transitions).
- $P_1(M, \mathcal{T}_O)$ holds if (i) $P_0(M, \mathcal{T}_O)$ holds, and (ii) there does not exist an unobserved path from s_0 to s_f.
- $P_{1'}(M, \mathcal{T}_O)$ holds if (i) $P_0(M, \mathcal{T}_O)$ holds, and (ii) there do not exist states s_1, $s_2 \in Q$ such that (a) there is an unobserved path from s_0 to s_2, (b) there is an unobserved path from s_1 to s_f, and (c) there is an unobserved path from s_1 to s_2. We refer to such a combination of nodes and edges as an unobserved reverse cyclic pattern between s_1 and s_2 (within M) .

For instance, on Fig. 2 with $\mathcal{T}_O' = \{e_2, e_6, e_9\}$, $P_0(\mathcal{T}_O')$ holds because $\mathcal{T}_O'$ is observable, $P_1(\mathcal{T}_O')$ holds because every path from s_0 to s_f uses at least one transition of $\mathcal{T}_O'$,

but $P_1(\mathcal{T}'_O)$ does not hold since there exists three non observable paths: e_4 from s_2 to s_3/ e_1e_3 from s_0 to s_3/ $e_5e_7e_8$ from s_2 to s_f.

By definition, $P_{1'}(M, \mathcal{T}_O) \Rightarrow P_1(M, \mathcal{T}_O) \Rightarrow P_0(M, \mathcal{T}_O)$, since for all s, there always exists a path from s to s. Let $\epsilon < 0 < 1 < 1'$. We define Best$(M, \mathcal{T}_O) = x \in \{\epsilon, 0, 1, 1'\}$ such that $P_x(M, \mathcal{T}_O)$ holds, but not $P_{xx}(M, \mathcal{T}_O)$ with $xx > x$, with the convention $P_\epsilon(M, \mathcal{T}_O)$ is always true. Informally, Best refers to the best properties a given set of transitions can ensure, if observed.

Proposition 2 *Let C be a component of M, and $\mathcal{T}_1, \mathcal{T}_2$ be subsets of transitions of C, respectively such that Best$(C, \mathcal{T}_1)$ = Best$(C, \mathcal{T}_2)$. Then, for all subset of transitions $\mathcal{T}_O$ of $M \setminus C$, we have Best$(M, \mathcal{T}_O \cup \mathcal{T}_1)$ = Best$(M, \mathcal{T}_O \cup \mathcal{T}_2)$.*

For $x \in \{0, 1, 1'\}$, we define $\mathcal{T}_x(M)$ as a smallest subset $\mathcal{T}_O$ of transitions of M such that $P_x(M, \mathcal{T}_O)$ holds. For a subset of transitions T of a component C of M, we also denote by $\mathcal{T}_x^{T,C}(M)$ a smallest set $\mathcal{T}_O$ such that $\mathcal{T}_O \cap C = T$ and $P_x(M, \mathcal{T}_O)$ holds. Every algorithm to compute the minimal observable set of transitions is recursive, taking the set of transitions considered observable as input. It is easy to modify them to input in the beginning not $\emptyset$ but T, and disallowing to select any new transitions in C, such that they compute $\mathcal{T}_x^{T,C}(M)$, and they do it faster than $\mathcal{T}_x(M)$ because they cannot choose among the transitions of C. As proved in proposition 2, the size of $\mathcal{T}_O$ is constant for several T such that $Best(C, T) = y$. If $|T'| > |T|$ with $Best(C, T) = Best(C, T')$, then $|\mathcal{T}_x^{T',C}(M)| > |\mathcal{T}_x^{T,C}(M)|$. We can use this idea to compute $\mathcal{T}_x(M)$ in a compositional manner, for a service M having component C:

MinimalDecomposition(M, C):

1. Compute a minimal set $\mathcal{T}_y(C)$ of transitions of C, $\forall y \in \{0, 1, 1'\}$.
2. Compute a minimal set $\mathcal{T}_0^{\mathcal{T}_y(C),C}(M)$ of transitions of M, $\forall y \in \{0, 1, 1'\}$.
3. Output a set of smallest size among $\mathcal{T}_0^{\mathcal{T}_y(C),C}(M)$.

For example, consider the service M having component C in Fig. 3.

1. A minimal set $\mathcal{T}_0(C) = \{(s'_0, s_2), (s_1, s'_f)\}$, $\mathcal{T}_1(C) = \{(s'_0, s_1), (s_2, s'_f)\}$, and $\mathcal{T}_{1'}(C) = \{(s'_0, s_2), (s_1, s'_f), (s_1, s_2)\}$.
2. The corresponding observable sets of M: $\mathcal{T}_0^{\mathcal{T}_0(C),C}(M) = \{(s'_0, s_2), (s_1, s'_f), (s_0, s_f)\}$ of size 3, $\mathcal{T}_0^{\mathcal{T}_1(C),C}(M) = \{(s'_0, s_1), (s_2, s'_f)\}$ of size 2, and $\mathcal{T}_0^{\mathcal{T}_{1'}(C),C}(M) = \{(s'_0, s_2), (s'_1, s_f), (s_1, s_2)\}$ of size 3.
3. $\mathcal{T}_0^{\mathcal{T}_1(C),C}(M)$ is a minimal observable set of M.

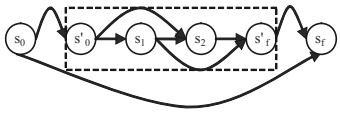


Figure 3. Service $M = (Q, s_0, s_f, \mathcal{T})$ having component $C = (Q', s'_0, s'_f, \mathcal{T}')$.

We can now state the main theorem of the paper.

Theorem 2 *Let $H = (M_i)_{i=1}^n$ be a hierarchical service. It is NP-complete in the size of H to compute $MO(\mathcal{H})$. Moreover, it takes at most time $O(\sum_{i=1}^n 2^{|M_i|})$.*

It is important to notice that since a service is in particular a hierarchical service (with hierarchy height of 1), we know that the problem is at least NP-hard. However, the complexity could be exponentially worse for hierarchical graphs, since a small hierarchical graph can represent an exponentially bigger flat graph. We prove that this is not the case. Moreover, we prove that the complexity is linear in the number of components, and exponential only in the size of each base component. That is, we prove that with a smart algorithm, one can compute efficiently the absolute minimal observable size even for huge hierarchical systems, as long as each component is small enough. The best case comparison is with respect to a hierarchical service of diameter $O(2^n)$, having n base components of size 2 (each one being reused 2^{n-1} times). The brute force non-compositional method runs on $\mathcal{H}$ and takes time $O(2^{2^n})$, while our method takes $O(n)$, that is a doubly exponential improvement (one exponential due to the reuse of components, and another due to decomposition).

6. Experimental Results

We tested our decomposition algorithm on hierarchical graphs. First, we choose a number (between one and nine) of base subcomponents in the graph. Then, we generate each of them randomly by using the *Synthetic DAG generation* tool (http://www.loria.fr/~suter/dags.html). We then generate inductively a hierarchical graph having these base subcomponents randomly using the same tool, by assigning two edges to these components. There is no reuse of components. For each value, we generate each hierarchical graph and each base subcomponent five times to compute the mean values (because of variation in runtime and

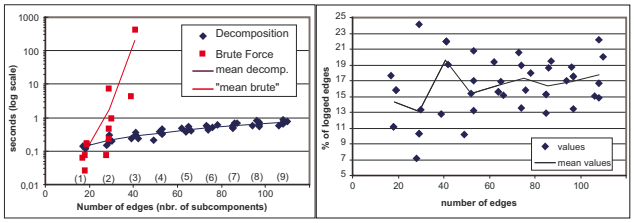


Figure 4. Execution time & observable size

observable size). We then unfold the hierarchical graphs as (flat) graphs, whose size is linear in the number of base components. We then run both a brute force algorithm and our hierarchical algorithm on these graphs. We do not input the hierarchical shape of the graph, instead the algorithm finds the optimal decomposition with a polynomial time algorithm, see [4]. Fig. 4(left) shows the times (in logarithmic scale) needed to compute a minimal observable set using brute force and our decomposition algorithm with respect to the number of edges (which is linear with respect to the number of base subcomponents). Our decomposition algorithm is indeed linear time with respect to the number of base subcomponents/number of edges (0.14s for an average number of edges of 18 and 0.73s for an average number of edges of 108), while the brute force is exponential in the number of edges, already timing out at a little over 40 edges. For 1 subcomponent, the overhead of our method makes the decomposition slightly worse than the brute force method. Fig. 4(right) shows the percentage of edges needed to be logged among all the edges. Both algorithms answer the same number on the same graphs but there is a huge variation among graphs, from one edge needs to be logged out of 4 to one edge out of 15. The mean value seems to tend to one out of 6.

7. Discussion and Conclusion

We studied compensation under partial log visibility. To the best of our knowledge, this problem has never been considered in the context of transactional services. We proposed a framework which uses the hierarchical nature of composite services, and gives an efficient algorithm to compute the absolute minimum number of transitions to observe in order to get compensability.

The algorithm we proposed considers only a subset of the whole set of transitions. It is thus straightforward to add constraints, such as, a subset of transitions "can/cannot be observed". It is very useful since in practice, we have to take into account privacy/security issues. The algorithm would then answer the absolute minimal observable set among those satisfying the constraints. Also, the hierarchical decomposition allows to deal with dynamicity. Indeed, if a service gets transformed (e.g., after the discovery/death of a sub-service), obtaining a minimal observable set would need recomputation, only at its level of the hierarchy (not below), plus *few* levels above (until the properties of a level are unchanged). It also allows to describe more accurately the details of a service which was considered atomic until now, in order to have feedback on where a service failed exactly. We explain in [4] how to deal with distributed systems, and with systems which are not given in a hierarchical way (using a folding algorithm).

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer Verlag, ISBN: 3540440089, 2004.

[2] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM TOPLAS, 23(3)*, pages 1–31, 2001.

[3] D. Biswas. Compensation in the world of web services composition. *In SWSWPC*, pages 69–80, 2004.

[4] D. Biswas and B. Genest. Minimal observability for transactional hierarchical services. *available at http://www.crans.org/genest/BG08.pdf*.

[5] R. Kumar and V. Garg. Modeling and control of logical discrete event systems. *Kluwer Academic*, 1995.

[6] F. Lin and W. Wonham. On observability of discrete-event systems. *Information Sciences, 44(3)*, pages 173–198, 1988.

[7] M. Lohrey and S. Manneth. The complexity of tree automata and xpath on grammar-compressed trees. *Theoretical Computer Science, 363(2)*, pages 196–210, 2006.

[8] S. Maheshwari. Traversal marker placement problems are np-complete. *Boulder Univ. Report CU-CS-092-76*, 1976.

[9] C. Ozveren and A. Wilsky. Observability of discrete event dynamical systems. *IEEE Trans. Auto. Control, 35(7)*, pages 797–806, 1990.

[10] W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. *In Jewels are Forever, Springer*, pages 262–272, 1999.

[11] K. Rohloff and J. Schuppen. Approximating the minimal cost sensor selection for discrete-event systems. *JDEDS, 16(1)*, pages 143–170, 2006.

[12] W. Sadiq and M. Orlowska. Analyzing process models using graph reduction techniques. *Inf. Syst., 25(2)*, pages 117–134, 2000.

[13] M. Sampath, R. Sengupta, S. Lafortune, K. Sinaamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Trans. Auto. Control, 40(9)*, pages 1555–1575, 1995.

[14] G. Weikum, A. Deacon, W. Schaad, and S. H. Open nested transactions in federated database systems. *IEEE Data Engg. Bulletin, 16(2)*, pages 4–7, 1993.

[15] A. Wombacher, P. Fankhauser, and E. Neuhold. Transforming bpel into annotated deterministic finite state automata for service discovery. *In ICWS*, pages 316–323, 2004.

[16] Web services transactions specifications. *http://msdn2.microsoft.com/en-us/library/ms951262.aspx*.

[17] T. Yoo and S. Lafortune. Np-completeness of sensor selection problems arising in partially observed discrete-event systems. *IEEE Trans. Auto. Control, 35(7)*, pages 797–806, 1990.

Using Boolean Cardinality Constraint for LTS Bounded Model Checking

Sachoun Park, Gihwon Kwon
Department of Computer Science, Kyonggi University
San 94-6, Yiui-Dong, Youngtong-Gu, Suwon-Si, Kyonggi-Do, Korea
{sachem, khkwon}@kgu.ac.kr

Abstract

Generally concurrent software is harder to find a bug than sequential one. Thus there have been a number of works on formal verification which exhaustively checks all behaviors of concurrent software. Bounded Model Checking (BMC) is widely used in formal verification of concurrent software systems and exhaustively checks whether some errors exist in execution traces of the given system or not within the given limit called as a bound.

In this paper, we develop the tool LTS-BMC that accepts both LTS (Labeled Transition System) as a modeling language and FLTL (Fluent Linear Temporal Logic) as a specification language. And the specification is checked against the model using a SAT solver. To translate them into a set of CNF clauses which is the input format to most SAT solvers, we propose efficient CNF encoding techniques and apply to several case studies. As a result, LTS-BMC shows a good performance.

Keywords : Formal Verification, Bounded Model checking, Labeled Transition System, Boolean Cardinality Constraint

1 Introduction

Model checking technique, which was designed for verifying hardware system in 1980′s, had made rapid progress through 1990′s with adoption of BDD(Binary Decision Diagram) as their inner data structure[1]. BDD based symbolic model checking tools were promptly disseminated in the field of system verification, also now a days, several verification tools based BDD are still widely used. But, that technique involves fatal problem, called state explosion. With changing the matter of concern of model checking from hardware to software verification, there were many active efforts to overcome the state explosion problem, so some studies make major accomplishment. For instance, in SLAM project, initiated by Microsoft to detect bugs of device driver, the framework of abstraction-verification-refinement was proposed and some real bugs were detected without state explosions[2,3].

Recent years, many researches, in which the model checking is considered as satisfiability problem, have made progress for alternative approach to alleviate the state explosion. These researches called BMC[4,5] (Bounded Model Checking) were based on high performance of SAT-Solver recently developed[6].

In this paper, in the line of previous studies, BMC for LTS(Labeled Transition System) is proposed because LTS is a proper model to describe concurrent system. Above all, how to encode a model into an input part of SAT-solver is precisely presented. FSP(Finite State Process), a textual language to express LTSs, was introduced for modeling and analyzing of behavioral flow of multi-threads of an java program[7]. In the LTSA(LTS Analyzer), a tool to support to analysis FSP, basically the explicit method is used for analyzing FSP, and the model checking method is trying lately[8], in which the automata theoretic approach was applied for LTL(Linear Temporal Logic) of LTS[9], but there is no approach to BMC for FSP yet. By the way, one research for BMC of LTS is accomplished by Toni[10,11]. Toni explained three different semantics for parallel composition of LTS in his Ph.D. thesis, and showed experimental results for verifying reachability, deadlock, and consistency properties. However the LTL bounded model checking for LTS was not described.

* This work was partly supported by the GRRC program of Gyeonggi province [2007-081-7, Context-aware Information Processing Technology for Effective Digital Contents Service] and the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (No. R01-2005-000-11120-0)

Due to the action-centered feature of LTS, to LTL model checking for LTS, the semantics of LTL is redefined by actions used in a corresponding to LTS. For model checking based on actions, a fluent, a sort of virtual state, must be defined and bounded model checking must be described based on the fluent. In the previously proposed fluent model checking, automata-theoretic approach was used, in that way firstly, given LTL formula was translated into Buchi automata, and then the automata was converted into property LTS, and finally, checking was performed on the composition between model LTS and property LTS[9]. But in this paper, this checking is performed by BMC and through experimental results with LTS-BMC we can check the model which cannot check with the existential LTS analyzing tool.

This paper is organized as follow. We overview BMC and define LTS and its parallel composition as a background in section 2. Then in section 3 the encoding, which translates given LTS model into the input part of a SAT-solver, is explained. Section 4 describes LTS-BMC tool we developed and analysis results of experiment and finally section 5 contains the conclusions and future works.

2 Background

2.1 BMC

The model checking exhaustively searches state space with breadth first method therefore during the model checking the amount of main memory become more exponential. So before finishing that the model checker searches for all state space, the main memory frequently is not available. But not all properties need to search for all state space. If the length of the counterexample of the property to be check is k, then the verification is finished within k-th iteration.

BMC has model, property, and bound k as inputs and then only performs verification within the bound. So BMC cannot provide information as to whether the property may be violated when more execution steps are taken. Therefore this technique is falsification, not verification.

In BMC, LTL formula is generally used to specify property. Both model and property are translated into CNF (Conjunctive Normal Form) formula, and SAT-Solver decides the satisfiability of the generated CNF formula. If the result of SAT solving is satisfable, then the property holds in the model, otherwise the property does not hold in the model. If the property holes however, then the error exists in the model, because the property means the error.

2.2 LTS

To model the behavior of components which work in concurrent systems LTS is used. LTS M consists of four tuples $\langle S, A, \Delta, q \rangle$:

- S is a non-empty finite set of states,
- A is a non-empty set of actions,
- $\Delta \subseteq S \times A \cup \{\tau\} \times S$ is the transition relation, the elements of which are called transitions of M and τ is silent action symbol which cannot be observed to the environment.
- $q \in S$ is an initial state of M.

By occurring of an action a, if the LTS $M = \langle S, A, \Delta, q \rangle$ transits other LTS $M' = \langle S, A, \Delta, q' \rangle$, that is represented by $M \xrightarrow{a} M'$ and $(q, a, q') \in \Delta$. Continuously occurring of actions makes the system to behave, so the sequence of actions is referred as to execution path or path.

In LTS M, the sequence of transitions generated by the sequence of actions $<a_1, a_2, \ldots, a_n>$ denotes $\sigma = <(s_1, a_1, s_2), (s_2, a_2, s_3), \ldots, (s_n, a_n, s_{n+1})>$, where $s_1 = q$, $\forall 1 \le i \le k. (s_i, a_i, s_{i+1}) \in \Delta$, and length of path $|\sigma|$ is n.

Let σ is a path on M, i-th step of the path becomes (s_i, a_i, s_{i+1}). If there is no outgoing transition in the state s_i, then that sates is called deadlock state and if the last state of a path is deadlock state, then the path is deadlock path.

For the set of LTS $\{M_1, \ldots, M_n\}$, their parallel composition is represented by $M_1 \,||\, \ldots \,||\, M_n$. Some action labeled in more than two LTSs synchronously occur in the composed LTS M and that action is called synchronized action, and the action labeled in specific LTS, the local action, occur by means of interleaved manner. Therefore at once also one action occur in composed LTS M. The definition of parallel composed LTS $M = M_1 \,||\, \ldots \,||\, M_n$ is as follow, when $M_i = \langle S_i, A_i, \Delta_i, q_i \rangle$.

- $S = S_1 \times \ldots \times S_n$,
- $A = A_1 \cup \ldots \cup A_n$,
- $\Delta = \{([s_1, \ldots, s_n], a, [t_1, \ldots, t_n]) \in S \times A - \{\tau\} \times S \mid \forall 1 \le i \le n.(a \in A_i \Rightarrow (s_i, a, t_i)) \vee (a \notin A_i \Rightarrow s_i = t_i))\} \cup \{([s_1, \ldots, s_n], \tau, [t_1, \ldots, t_n]) \in S \times \{\tau\} \times S \mid \exists 1 \le i \le n. (s_i, \tau, t_i) \in \Delta_i \wedge \forall 1 \le j \le n. i \ne j \Rightarrow s_i = t_i \}$,
- $q = [q_1, \ldots, q_n]$.

The state in composed LTS by the number of n is expressed by n-tuple but the set of actions in composed

LTS is the union of each A_i, so that the element of A is called global action. If a global action occurs, then LTS that has the corresponding to local action can move but LTS that never have the same local action is idle. If the silent action τ occurs, then the local LTS have the silent action can transit.

3 Encoding using BCC

3.1 BCC

Before we describe encoding of LTS model, we explain the BCC(Boolean Cardinality Constrants) frequently used in the encoding of our model and predicates. Given a set of Boolean variables $X = \{x_1, \ldots, x_n\}$, BCC are formulae expressing that at most (resp. at least) k out of n variables are true. If selecting at least one element among X denotes *AtLeastOne(X)*, then the formula is expressed as follow.

$$AtLeastOne(X) = \bigvee_{i=1}^{n} x_i$$

Similarly selecting at most one among X is as below.

$$AtMostOne(X) = \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \left(\neg x_i \vee \neg x_j\right)$$

Therefore the predicate of selecting exactly one element among X, *ExactlyOne(X)* is *AtMostOne(X)* $\wedge$ *AtLeastOne(X)*. However, above encoding method is naïve and more efficient encoding ways are introduced by Bailleux[12], Sinz[13], and Kwon[14]. The BCC encoding of Bailleux is efficient for selecting exactly k out of n variables and he uses binary tree and unary representation of an integer value. Sinz's method is optimal for selecting at most k among n variables and takes advantage of a sequential counter for encoding BCC into CNF. And Kwon proposed an encoding method that has best performance in selecting one out of n variables and he adopted the concept of hierarchies among Boolean variables. We experiment with these encoding methods and compare with them in our tool.

3.2 Mode Encoding

Now we will explain some predicates used in definition of encoding of LTS. Below step i is i-th step on the execution path.

- *Select(s, i)* is true if state s is selected in the i-th step.
- *Select (a, i)* is true if action a is selected in the i-th step.
- *Enable(s, Act, i)* is true if in the i-th step set of actions *Act* is enable in state s. when $Act = \{a_1, \ldots, a_n\}$, $Enable(s, Act, i) = Select(s, i) \Rightarrow Select(a_1, i) \vee \ldots \vee Select(a_n, i)$.
- *Disable(Act, i)* is true if in the i-th step set of actions *Act* is disenable, is equal to the formula $\forall a \in A. \neg Select(a, i)$.

Already mentioned LTS $M = \langle S, A, \Delta, q \rangle$ consists of a set of states, a set of actions, transition relations, and an initial state. Therefore encoding of M can be split into encoding of each component in i-th step. First, the formula the corresponding to the set of states is (1).

$$ExactlyOne(S, i) \quad (1)$$

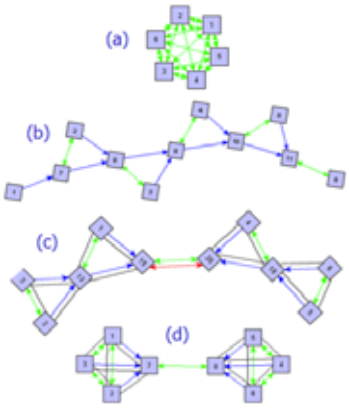


Figure 2. At most one of six: (a) Naïve method, (b) Sinz's, method (c) Bailleux' method, (d) Kwon's method

Only one state is selected at each step of the execution path, so the predicate (1) which expresses to select one state in i-th step is defined by extending the predicate *ExactlyOne(S)*. And more detail, given a set of states $S = \{s_1, \ldots, s_n\}$, *ExactlyOne(S, i)* is equal to the predicate $ExactlyOne(\{Select(s_1, i), \ldots, Select(s_n, i)\})$. Similarly the extended predicate for the set of actions can be defined. Because only one global action may occur in M, the predicate about selecting action in i-th step is denoted by (2).

$$AtLeastOne(A, i) \quad (2)$$

Also given a set of actions $\{a_1, \ldots, a_n\}$, *AtLeastOne(A, i)* is equal to $AtLeastOne(\{Select(a_1, i), \ldots, Select(a_n, i)\})$. Encoding about transition relations needs two formulae, that is, the condition for occurring transition and the condition after the transition occurring. In each state s, when step-i, the predicate expressing the condition for occurring transition is (3).

$$\forall s.Enable(s, Act, i) \quad (3)$$

The condition of enable transition is when a state is selected the possible action to occur in s is selected. Thus occurring transition is represented by predicate express

possible actions to occur in each state. And then for all transition relations $(s, a, t) \in \Delta$ in i-th step, the formula of the condition after the transition occurring is defined by (4).

$$Trans(\Delta, i) = \forall (s, a, t) \in \Delta.\ Select(s, i) \wedge Select(a, i) \Rightarrow Select(t, i+1) \quad (4)$$

Because the initial state of M is selected in step zero, we can use below predicate about q in M. where q represents the Boolean variable of the initial state.

$$Select(q, 0) \quad (5)$$

Thus given a bound k, the formula to encode *LTS M* *Encode(M, k)* is denoted (6), conjunction from (1) to (5). About encoding for the initial state, because of exactly one state can be selected and *Select(q, 0)*, in step *0*, it guarantees that other states is not selected except q.

$$\begin{aligned} Encode(M, k) = {} & Select(q, 0) \\ \wedge\ & \forall 0 \le i < k.(ExactlyOne(S, i) \\ & \wedge AtMostOne(A, i) \\ & \wedge \forall s.\ Enable(s, Act, i) \\ & \wedge Trans(\Delta, i)) \end{aligned} \quad (6)$$

Universal quantifier means $\wedge$-operation of all instances, so the final formula of the model (6) is kept up CNF. The formula of the deadlock property can be expressed by $\exists 0 \le i < k.Disable(A, i)$ which means that there is the state in which any actions can be occurred.

3.3 Encoding a parallel composed LTS

There are two approaches for encoding of the parallel composed LTS. One approach is that we first generate the composed model by means of explicit manner and then encode the composed model, and another one is that we first encode each local LTS, and next compose. First approach is finished by encoding for the composed LTS *M*, *Encode(M, k)*. And second approach is that interleaving semantics among local models is encoded with encoding for each local model. The brief description of the second approach is as follow.

- States, actions, initial state, and enable actions in the particular state are encoded by the same method of the local LTS.
- Variables for interleaving and global actions are declared and the connection between global actions and local actions are encoded with interleaving variables.
- Transitions in each LTS are encoded with the occurrence of global actions.

In previous chapter as we described, initial state of composed *LTS* $M = M_1 \,||\, ... \,||\, M_n$ is $[q_1, ..., q_n]$, so we can encode like below (7).

$$\forall 1 \le j \le n.Select(q_j, 0) \quad (7)$$

For states, actions, and enable actions of each LTS, in i-step, we can encode such as (8), (9), (10).

$$\forall 1 \le j \le n.ExactlyOne(S_j, i) \quad (8)$$

$$\forall 1 \le j \le n.AtMostOne(A_j, i) \quad (9)$$

$$\forall 1 \le j \le n.\ \forall s \in S_j.\ Enable(s, Act, i) \quad (10)$$

The set of global actions $G = \{g_1, ..., g_m\}$ is encoded like the set of local actions as the formula (11). And the formula (12) is the encoding for the set of variables for interleaving semantics $IN = \{in_1, ..., in_n\}$, which is encoded by selecting exactly one of n, because only one LTS must be selected.

$$AtMostOne(G, i) \quad (11)$$

$$ExactlyOne(IN, i) \quad (12)$$

The set G is corresponding to the set of actions of the parallel composed LTS. For the connection between global and local action, let the $MA(g) = \{a \mid \exists 1 \le j \le n.\ a \in A_j \wedge a = g\}$is the set of actions, which are corresponding to a global action g, in each local LTS and $ING(g) = \{in_j \in IN \mid g \in A_j\}$. At i-th step, the encoding of each global action is (13).

$$\begin{aligned} Matching(i) = {} & \forall g \in G.\ \forall a \in MA(g).Select(g, i) \\ & \Leftrightarrow Select(a, i) \\ & \wedge \exists_1 in \in ING(g).Select(in, i) \end{aligned} \quad (13)$$

Finally, encodings for transition relations of each LTS $M = \langle S, A, \Delta, q \rangle$ at i-th step are formulated by (14) and (15). The formula (14) is the constraint for transit, and (15) is the constraint for idle, where $s \in S,\ a \in A,\ (s, a, t) \in \Delta$, and $g \in G$.

$$Select(s, i) \wedge Select(a, i) \wedge Select(g, i) \Rightarrow Select(t, i+1) \quad (14)$$

$$Select(s, i) \wedge Select(a, i) \wedge \neg Select(g, i) \Rightarrow Select(s, i+1) \quad (15)$$

Therefore the formula made by conjunction from (7) to (15) is encodings for a given parallel composed LTSs.

3.4 Encoding property

Property is generally represented by LTL formula which is defined by follow.

$$\phi ::= Fl \mid \neg Fl \mid \phi \vee \psi \mid \phi \wedge \psi \mid X\phi \mid F\phi \mid G\phi \mid \phi U \psi.$$

Fl means fluent, which is defined by a pair of sets, a set of initiating actions I_{Fl} and a set of terminating action T_{Fl}.

$$Fl \equiv \langle I_{Fl}, T_{Fl} \rangle Iintially_{Fl}$$

where $I_{Fl}, T_{Fl} \subset G$ and $I_{Fl} \cap T_{Fl} = \varnothing$ and a fluent Fl may initially be true or false at step 0 as denoted by the attribute $Iintially_{Fl}$. Therefore a fluent is a kind of state which value is true or false according to the occurrence of actions. $E_{Fl} \subset G$ is also actions and neither initiating actions nor terminating actions. For the occurrence of E_{Fl}, the value of fluent is preserved. Encoding fluent is as follow, and for the rest LTL formula, we follow [15,16].

$$\begin{aligned} Encoding(Fl) = Iintially_{Fl} & \\ \wedge I_{Fl} \Rightarrow Fl & \\ \wedge T_{Fl} \Rightarrow \neg Fl & \\ \wedge E_{Fl} \wedge Fl \Rightarrow Fl & \\ \wedge E_{Fl} \wedge \neg Fl \Rightarrow \neg Fl & \end{aligned}$$

4. Experiments

The LTS-BMC tool we developed is consists of three phases as figure 2. In the parsing phase, the input is FSP file and the result is LTSs. And in the second phase, CNF formulae of model and property are encoded. Finally decision whether generated formulae is satisfiable or not is performed by the state-of-the-art SAT-solver, minisat[17]. If the result of solver is unsatisfiable, then the bound k is incremented, and if the result is satisfiable then because this case means to fine an error, which is a counterexample, it resolves the error traces by decoder. Although after increasing k enough, the result still remain unsatisfiable, it is regarded as to fail to find the counterexample and solving is terminated.

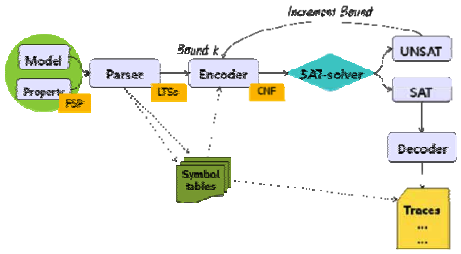


Figure 2. Overview of LTS-BMC

To compare encoding methods, we experiment with the example appeared in [18]. The concepts of atomic operation and interleaved execution of concurrent programs are conveniently demonstrated by a program with two processes, each of executes K iterations of the statement $N := N + 1$, where N is a shared variable with an initial zero value. Intuitively, in the final state $K \leq N \leq 2K$; [18] described a scenario whose final state is $N = 2$. In our experiments, $K = 3$ and the number of processes are increased one by one. The FSP code in figure 3 is the employed example in the case of two processes.

```
bound 20
const Imax = 3
range Int = 0..Imax

VAR(Init = 0) = VAR[Init],
VAR[v:Int] =   (read[v] ->VAR[v]
               |print v]->VAR[v]
               |write c:Int]->VAR[c]).

Proc = Proc[0],
Proc[i:Int] = ( when(i < Imax)
                    read[j:0..Imax-1] ->
                    inc ->
                   write[j+1] -> Proc[i+1]
              | when(i == Imax)
                    end -> END)+{write[0]}.

||C = ({a,b}::VAR || a:Proc || b:Proc)/
                              {end/{a,b}.end}.
Test = (end -> {a,b}.print[2] ->STOP).
||C_Test = (C || Test).

fluent F = <{{a,b}.print[2]}>
assert A = <> F
```

Figure 3. FSP code for the extreme interleavings

Table 1 shows experimental results by means of LTSA and LTS-BMC. LT-SEQ stands for that we use Sinz's method[13] as encoding BCC in LTS-BMC. It was performed on Windows XP with a 1Gb memory and 1.86GHz CPU and the timeout is 1000sec. the symbol "-" means the experiment to be failed caused by memory overflow. #P is the number of process, #state is the number of states, #trans. is the number of transitions, k is the bound used in BMC, #var. and #clause are the number of variables and clauses in CNF, respectively.

<Table 1> Experimental results

#P	LTSA			LTS-BMC(LT-SEQ)			
	#state	#trans.	time	k	#var.	#clause	time
2	272	842	0.11	20	5,523	16,882	0.32
3	7,540	34,712	3.68	29	11,520	35,771	6.18
4	170,425	1,130,530	6.91	38	19,695	61,662	146.59
5	-	-	-	47	30,624	95,699	780.78
6	-	-	-	56	42,579	134,450	286,25

In the table 1, although the solving time of LTS-BMC is slower than one of LTSA, we recognize state spaces of LTSA are growing exponentially in contrast of the linear increasing of the size of CNF formulas generated by LTS-BMC. So LTS-BMC has more benefit than LTSA when the number of processes is growing. However the solving time of LTS-BMC must be improved. We consider the formula (10) described in section 3.3 as one of main obstacle. Below figures are consumed memory and time for experiments with several BCC methods. Commander indicates Kwon's method[14] and UT_MGAC indicates Bailleux's method[12]. In this experiment LT-SEQ shows the best performance among several BCC methods.

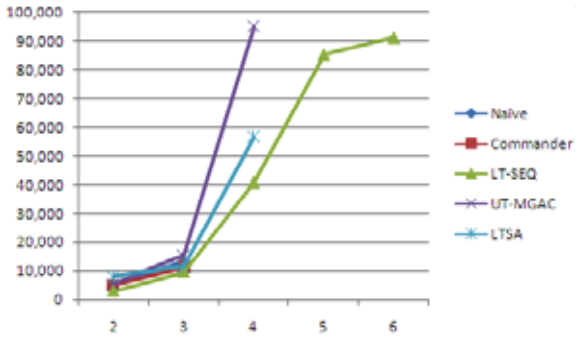


Figure 4. Amount of the memory used in solving

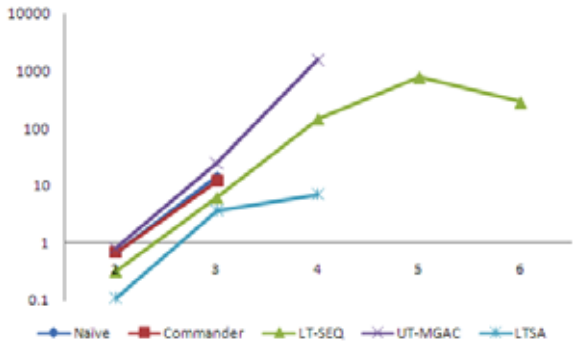


Figure 5. Sloving times

5 Conclusions

To overcome the limitation of the model checking technique, the problem of state explosion, the research about BMC is active. In this paper, we developed the BMC tool for LTS, which is a proper model to describe concurrent system, and experimented with interleaving models. Through experiment, we showed verifying LTS model which could not deal with an LTSA. Remained works are improving the encoding method of model and developing unbounded model checking for LTS.

References

[1] E. M. Clarke and E. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logics," In Logic of Programs: Workshop, volumn 131 of LNCS, pages 52-71. Springer-Verlag, 1981.

[2] J. P. Quielle and J. Sifakis, "Specification and verification of concurrents systems in CESAR", In Proceedings of the 5th International Symposium of Programming, pages 337-350, 1981.

[3] http://research.microsoft.com/slam/

[4] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs", In Proceeding of Workshop on Tools and Algorithms for the Construction and Analysis of Systems, LNCS, Springer-Verlag, 1999.

[5] A. Biere, A. Cimatti, E. Clarke, Ofer Strichman, and Y. Zhu, "Bounded Model Checking", Vol. 58 of Advances in Computers, 2003. Academic Press (pre-print).

[6] M.W. Moskewicz, C. Madigan, Y. Zhao, L. Zhang and S. Malik, "Chaff: Engineering an Efficient SAT Solver," In Proceedings of Design Automation Conference, 2001.

[7] J. Magee and J. Kramer, Concurrency – State Models and Java Programs, Chichester, John Wiley & Sons, 1999.

[8] http://www.doc.ic.ca.uk/ltsa/

[9] D. Gannakopoulou and J. Magee, "Fluent Model Checking for Event-based Systems," In Proceedings of ESEC/FSE03, 2003.

[10] T. Jussila, "BMC via dynamic atomicity analysis," In Proceedings of the International Conference on Application of Concurrency to System Design, IEEE Computer Society, June 2004.

[11] T. Jussila, K. Heljanko, and I. Niemela, "BMC via on-the-fly determinization," In Proceedings of the 1st International Workshop on Bounded Model Checking, 2003.

[12] O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of Boolean cardinality constraints," In Proceedings of the CP 2003, vol. 2833, LNCS, 2003.

[13] C. Sinz, "Towards an optimal CNF encoding of Boolean cardinality constraints," In Proceedings of the CP 2005, vol. 3709, LNCS, 2005.

[14] G. Kwon and W. Klieber, "Efficient CNF Encoding for Selecting 1 from N Objects," In the Proceeding of CFV07, 2007.

[15] A. Cimatti, M. Pistore, M. Roveri, and R. Sebastiani, "Improving the Encoding of LTL Model Checking into SAT," In the Proceeding of 3rd VMCAI, vol. 2294, LNCS, 2002.

[16] T. Latvala, A. Bere, K. Heljanko, and T. junttila, "Simple Bounded LTL Model Checking," In proceedings of the 5th VMCAI, vol. 2937, LNCS, 2004.

[17] N. Een and N. Sorensson, "An Extensible SAT-solver," In the Proceedings of SAT 2005, vol. 2919, LNCS, 2005.

[18] Mordechai Ben-Ari , Alan Burns, Extreme Interleavings, IEEE Concurrency, v.6 n.3, p.90-91, July 1998.

Japanese Puzzle as a SAT Problem

Sachoun Park, Gihwon Kwon
Department of Computer Science, Kyonggi University
San 94-6, Yiui-Dong, Youngtong-Gu, Suwon-Si, Kyonggi-Do, Korea
{sachem, khkwon}@kgu.ac.kr

Abstract

In this paper, we are interested in automatically solving Japanese puzzle which is also known as Nonogram and aims to reveal a hidden picture according to numbers given at the side of the two-dimensional board. For example, the numbers "4 3" mean there are sets of four and three consecutive filled squares, in that order, with at least one blank square between successive groups.

To regard it as a SAT problem, Boolean Cardinality Constraint (BCC) is seemed to be a natural choice to represent as a set of CNF clauses. However, BCC alone is not enough to identify a sequence of n filled squares given the number n. To select a sequence of filled squares, we add the adjacency constraint to traditional BCC and applied this idea to a number of this kind of puzzles. As a result, our idea shows much better performance than when BCC alone is used.

Keywords : Adjacency, Boolean cardinality constraints, CNF encoding, Satisfiability, Solving puzzle

1 Introduction

After the DPLL[1] was proposed for the algorithm of SAT solving in 60's, in the last decade the performance of SAT solvers was highly improved[2,3,4]. Due to these improvements, in many application areas such as AI planning[5], hardware and software verification[6,7], solving puzzles[8], and reconstruction of images[9], problems are regarded as SAT problems. To deal with problems via SAT, those are specified by several constraints and then encoded to conjunctive normal form (CNF). CNF is the conjunction of clauses composed of disjunction of literals that is an atomic proposition or the negation of an atomic proposition.

However, many types of constraints that appear in practical problems have no natural expression in the CNF. Boolean cardinality constraint, which expresses bounds of the number of variables to be true in the set of Boolean variables, is one of these. So far, there are many researches about CNF encoding of BCC, which are concentrated on generating efficient CNF formulae. Here, the efficiency means that generated formulae must be kept relatively small number of variables and clauses with respect to given variables and must be made the solving time shorter. Sinz applied sequential counter into CNF encoding for BCC in [10], and Bailleux and Boufkhad, in [9], used binary tree whose nodes function as the middle summation. When n is the number of given Boolean variables and k is the number of variables to be true, Sinz's and Bailleux's methods need $O(n \cdot k)$, $O(n^2)$ clauses and $O(n \cdot k)$, $O(n \cdot log\, n)$ auxiliary variables, respectively.

Another approach showing the efficiency of encoding technique is testing it on some applications. In [9], Bailleux and Boufkhad had tested their technique on a problem arising in 2-D discrete tomography which is the reconstruction of a 2-D grid, given its projections in four directions. From the problem we could get an idea about the adjacency constraint (ADC) and define its CNF encoding. So given projection in two direction with adjacency information, we could reconstruct the image of a 2-D grid. In fact, this method is close to solve Japanese puzzle(Nonogram)[11,12,13]. To express the ADC, we assume the sequence of given Boolean variables and we say that variables assigned true are adjacent when all those are neighbored with each other in the sequence.

Through experiments we prove the applicability of the ADC. To do this, we define the naïve encoding of ADC and then we explain how to combine it within BCC to translate problems of reconstructing image to CNF formulae. Also we show the method of applying ADC to BCC using binary tree and sequential counter.

This paper is organized as follow. As background CNF encodings for BCC is described in section 2, where BCC

* This work was partly supported by the GRRC program of Gyeonggi province [2007-081-7, Context-aware Information Processing Technology for Effective Digital Contents Service] and the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (No. R01-2005-000-11120-0)

is described by the constraint $l \leq |E'| \leq m$, which means that the number of set of Boolean variables assigned true E' in given Boolean set E is between l and m. And the description parts into three subsection: naïve encoding, encoding using a binary tree, and encoding using a sequential counter. Then in the section 3, we propose the concept of adjacency constraint and define CNF encoding for ADC in those three parts. In section 4, we present experimental results on three types of examples with four different methods. Finally we conclude in section 5.

2 Backgrounds

2.1 Naïve encoding

In this subsection, we explain naïve encoding for BCC. Let $E = \{e_1, \dots, e_n\}$ is a set of Boolean variables and $E' \subseteq E$ represents a set of variables to be true. To translate BCC for $l \leq |E'| \leq m$ into CNF formulae, we must separate the constraint into two parts $l \leq |E'|$ and $|E'| \leq m$, and translate each part of the constraint into CNF, and then make the conjunction of two formulae. These two constraints $l \leq |E'|$ and $|E'| \leq m$ mean that the size of E' is at least l and at most m, respectively. For all subsets of E, $E_1', \dots, E_{\binom{n}{l}}'$, where each length of subsets is l, the naïve way of converting the constraint $l \leq |E'|$ to CNF is as follow.

$$\bigvee_{1 \leq i \leq \binom{n}{l}} \bigwedge_{1 \leq j \leq l} e_j \in E_i' \tag{1}$$

If the size of a subset $E' \subseteq E$ to be true is at most m, then we can surely exclude variables included in $E - E'$. Therefore, all subsets of E, $E_1'', \dots, E_{\binom{n}{m+1}}''$, where each length of subsets is $m+1$, the second constraint $|E'| \leq m$ is expressed by follow.

$$\bigwedge_{1 \leq i \leq \binom{n}{m+1}} \bigvee_{1 \leq j \leq m+1} \neg e_j \in E_i'' \tag{2}$$

Thus for the problem of selecting exactly k elements in the set $E = \{e_1, \dots, e_n\}$, we can generate corresponding propositional formulae as converting $k \leq |E'| \leq k$ to CNF. In the case of the constraint $|E'| \leq k$, it requires $\binom{n}{k+1}$ clauses of length $k+1$. In the worst case of $k = \lceil n/2 \rceil - 1$, the number of generated clauses is $O(2^n/\sqrt{n/2})$, so next subsections present better encodings for $l \leq |E'| \leq m$.

2.2 Encoding using a binary tree

The encoding method explained in this subsection was proposed by Bailleux and Boufkhad[9]. The encoding uses a unary representation of integers, so to express the number of variables assigned true, it must use auxiliary variables. For example, if we want to present the value of an integer k such that $0 \leq k \leq n$ with Boolean variables $s_1, s_2, \dots, s_n$, then it becomes $s_1 \wedge \dots \wedge s_k \wedge \neg s_{k+1} \wedge \dots \wedge \neg s_n$. Thus when the value of k is in the bound $l \leq k \leq m$, we can generally express it as follow formula.

$$\bigwedge_{1 \leq i \leq l} s_i \quad \wedge \quad \bigwedge_{m+1 \leq j \leq n} \neg s_j \tag{3}$$

For example, given the set of Boolean variables $E = \{e_1, e_2, e_3, e_4, e_5\}$, to convert the BCC to CNF using the unary representation, it requires 5 auxiliary variables. Let the set of auxiliary variables is $S = \{s_1, s_2, s_3, s_4, s_5\}$, if variables to be true in E are only e_1 and e_4, then for the number 2, it could be expressed by the formula $s_1 \wedge s_2 \wedge \neg s_3 \wedge \neg s_4 \wedge \neg s_5$. Therefore, to represent all cases of allowing variables to be true in E as unary representation of S, the set S is allocated to the root node of the tree, each group of auxiliary variables $(l_1, l_2), (r_1, r_2, r_3), (r_4, r_5)$ is allocated to each non-terminal nodes in the tree like figure 1, and each variable in E is allocated to a leaf node in a one-to-one way, thus all nodes in the tree represent the sum of the sub-tree whose root is that node.

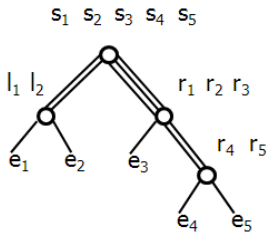


Figure 1. Binary tree and assigning auxiliary variables

As the previous example, if variables allowed to be true are only e_1 and e_4, then each variables located by non-terminal node is converted to $l_1 \wedge \neg l_2$, $r_4 \wedge \neg r_5$, and $r_1 \wedge \neg r_2 \wedge \neg r_3$, respectively, which are correspond to their sub-sum. Consequently, to convert $l \leq |E'| \leq m$ to CNF using a binary tree, it start from a isolated node located by variables for the integer n and proceed iteratively: to each terminal node which has $n' > 1$ variables, we connect two children to be located by $\left\lfloor \frac{n'}{2} \right\rfloor$, $n' - \left\lfloor \frac{n'}{2} \right\rfloor$ variables, respectively. This procedure builds a binary tree with n leaves assigned by $e_i \in E$. In this binary tree, when $A = \{a_1, \dots, a_p\}$ is a set to assigned in a non-terminal node and $L = \{l_1, \dots, l_{p_1}\}$ and $R = \{r_1, \dots, r_{p_2}\}$ are variables assigned in left and right child of the node, the following CNF formula expresses their relation, where $a_0 = l_0 = r_0 = 1$ and $a_{p+1} = l_{p_1+1} = r_{p_2+1} = 0$.

$$\bigwedge_{0\le i\le p_1}\bigwedge_{0\le j\le p_2}\left(l_i \wedge r_j \Rightarrow a_{i+j}\right) \wedge \left(a_{i+j+1} \Rightarrow l_{i+1} \vee r_{j+1}\right) \quad (4)$$

According to our observation, the conjunction of (3) and the left part of the above formula is the CNF representation of $l \le |E'|$ constraint and the conjunction of (3) and the right part is for $|E'| \le m$. Therefore, if we apply formula (4) for all non-terminal nodes and conjoin them with (3), we can represent $l \le |E'| \le m$. As I mentioned in section 1, this method requires $O(n \cdot log\ n)$ clauses and $O(n^2)$ variables.

2.3 Encoding using a sequential counter

In this section, we explain the translation method using a sequential counter proposed by Sinz[10]. Let input variables are $E = \{e_1, \dots, e_n\}$. A sequential counter circuit computes partial sums $s_i = \sum_{j=1}^{i} e_j$ for increasing values of i up to the final $i = n$, and the values of all s_i's are represented as unary numbers. Sub-circuit is computing a partial sum s_i in unary representation and the maximal value of the partial sum is m. Therefore, it requires $(n-1)\cdot m$ auxiliary variables.

To convert this circuit to CNF, we define equations for the partial sum $s_{i,j}$ such as formula (5). In [10], to convert this circuit to CNF, implication operator is used at (a), (b), and (c) in (5). This way could optimize the CNF encoding of the constraint $|E'| \le m$, but if we use equivalence operator instead of implication, then remaining relatively small number of clauses the constraint $l \le |E'| \le m$ can be represented.

If we use equivalence operator instead of implication at (a), (b), and (c) in formula (5), then the constraint $l \le |E'|$ is represented by the formula (6).

$$\begin{aligned}
&e_1 \Rightarrow s_{1,1}\ \text{(a)}\\
&\text{for } 1 < j \le m\\
&\quad \neg s_{1,j}\\
&\text{for } 1 < i < n\\
&\quad (e_i \vee s_{i-1,1}) \Rightarrow s_{i,1}\ \text{(b)}\\
&\quad \text{for } 1 < j \le m\\
&\qquad ((e_i \wedge s_{i-1,j-1}) \vee s_{i-1,j}) \Rightarrow s_{i,j}\ \text{(c)}\\
&e_i \Rightarrow \neg s_{i-1,m}
\end{aligned} \quad (5)$$

$$s_{l,l} \vee \dots \vee s_{n-1,l} \vee (e_n \wedge s_{n-1,l-1}) \quad (6)$$

Therefore, the CNF formulae encoding of $l \le |E'| \le m$ consist of $4m \cdot n - 7m + 4$ clauses. Although the number of clauses of CNF formula for $l \le |E'| \le m$ with equivalence operator is larger than CNF formula for $|E'| \le m$ requiring $2m \cdot n + n - 3m - 1$ clauses, we could translate $l \le |E'|$ into CNF simply and the increasing rate of whole formulae is still linear.

3 CNF encoding for Adjacency constraint

To express adjacency, we consider the sequence of Boolean variables, so we regard the set of Boolean variables E used in the previous section as a sequence of Boolean variables.

3.1 Naïve encoding for ADC

To present adjacency of variables, we must assume the order between those variables. For example, if we want to represent to allow adjacent two variables to be true out of five Boolean variables $E = \{e_1, \dots, e_5\}$, then we can think about two different type of adjacency. If the third variable is the one of two variables assigned by true, then we can say that either the second variable or the forth variable will be true, or at least the first and the fifth variables will be false; the former for the positive representation, the latter for the negative representation. Firstly, the positive representation of ADC is a formula to fulfill that the exactly one out of ($e_1 \wedge e_2$), ($e_2 \wedge e_3$), ($e_3 \wedge e_4$), and ($e_4 \wedge e_5$) comes true. More generally, in a set of Boolean variables $E = \{e_1, \dots, e_n\}$, the CNF formula of the representation of a adjacent k variables is expressed by (7). In the formula (7), the function $ExactlyOne(\cdot)$ receives a set of Boolean variables as a parameter and returns a CNF formula to express the exactly one of them to be true. From (7), CNF formula is generated with $n-k+1$ auxiliary variables and $(n-k+1)\cdot(k+1)+\alpha$ clauses, where the α corresponds to that function.

$$\bigwedge_{i=1}^{n-k+1}\left(a_i \Leftrightarrow \bigwedge_{j=i}^{i+k-1} e_j\right) \wedge ExactlyOne(\{a_1, \dots, a_{n-k+1}\}) \quad (7)$$

Secondly, the negative representation of ADC for the above example is expressed like the formula ($e_1 \Rightarrow \neg e_3 \wedge \neg e_4 \wedge \neg e_5$) $\wedge$ ($e_2 \Rightarrow \neg e_4 \wedge \neg e_5$) $\wedge$ ($e_3 \Rightarrow \neg e_5$) and in this formula any auxiliary variables are not needed. For a set of Boolean variables $E = \{e_1, \dots, e_n\}$, the negative representation of a adjacent k variables is identical to the blow formula (8), and it generates $\frac{(n-k+1)\cdot(n-k)}{2}$ clauses.

$$\bigwedge_{i=1}^{n-k}\bigwedge_{j=i+k}^{n} e_i \Rightarrow \neg e_j \quad (8)$$

So far, we explained negative and positive representation for ADC, but on applying the ADC, it must be used together with the formula for BCC, that is, the BCC for the number of variables allowed to be true and the ADC for the constraint that those variables must be adjacent each other. However, the positive representation of ADC belongs to the constraint of $k \leq |E'|$ because all of adjacent k variables are included among the $\binom{n}{k}$ combination using (1). Consequently, as the positive representation combined with the constraint $|E'| \leq k$, it can be formulated that exactly adjacent k variables become true.

Similarly, the negative representation of ADC belongs to the constraint of $|E'| \leq k$ because the formula (8) implies the formula (2). Consequently, as the negative representation combined with the constraint $k \leq |E'|$, it can be formulated that exactly adjacent k variables become true. In the next subsections, we will describe how to apply the ADC into methods using binary tree and sequential counter in the viewpoint of the negative representation.

3.2 Encoding of ADC using a binary tree

If we adopt BCC using a binary tree to express ADC, we can apply the advantage of structure of the tree. For example, if we want to restrict adjacent two variables to be true out of five Boolean variables, then the formula of ADC becomes $(e_1 \Rightarrow \neg r_1) \wedge (e_2 \Rightarrow \neg r_4) \wedge (e_3 \Rightarrow \neg e_5)$ because according to the formula (4) $\neg r_1$ and $\neg r_4$ are identical to $\neg e_3 \wedge \neg e_4 \wedge \neg e_5$, $\neg e_4 \wedge \neg e_5$, respectively. Therefore, when $E = \{e_1, \dots, e_n\}$ is a set of Boolean variables, CNF encoding for ADC using a binary tree is formulated by follow.

$$\bigwedge_{i=1}^{n-k} \bigwedge_{a \in LCA(i)} e_i \Rightarrow \neg r_1^{right(a)} \tag{9}$$

$LCA(i) = \{lca(i,j) | i + k \leq j \leq n\}$ presents the set of the indices of non-terminal node for e_i, where $lca: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ is the function that from indices of two nodes returns the index of the least common ancestor of them. Also $right: \mathbb{N} \rightarrow \mathbb{N}$ is the function that from the index of a non-terminal node returns the index of the right child of it. Thus $r_1^{right(a)}$ means the first assigned variable in the non-terminal node whose index is a. Consequently the conjunction of (3), (4), and (9) is CNF encoding of ADC using a binary tree, which requires no auxiliary variables and $(n-k) \cdot log\, n$ clauses in the worst case. We will say this method with TA which means the adjacency in a tree.

As we mentioned in the previous subsection, the left part(TL) and the right part(TR) of the formula (4) corresponds to the constraint $k \leq |E'|$, $|E'| \leq k$, respectively. So, TL$\wedge$(8) and TR$\wedge$(7) are available for another encoding methods using binary tree.

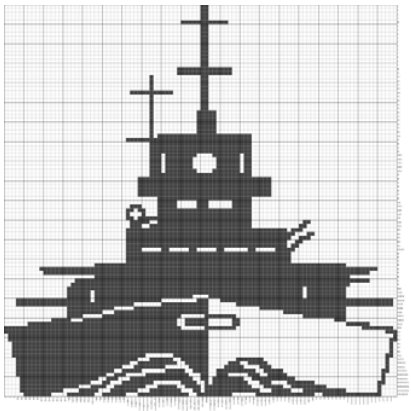


Figure 2. An example of 100×100 Japanese puzzle

3.3 Encoding of ADC using a sequential counter

There are four different methods in encoding of ADC using a sequential counter. SC is referred to as the formula (5) which is optimized to represent $|E'| \leq m$. While SC means that the number of variables to be true is at most m, if we consider that the number of variables to be false is at most $(n-l)$, also the constraint $l \leq |E'|$ is represented with a sequential counter. To do this, we add negation operator to each input variable in a sequential counter and for use $(n-l)$ instead of the parameter l. This method is called SC^{-1}. And the method using equivalence at (a), (b), and (c) in (5) is called SC^{M}.

Therefore, the first method expressing ADC with a sequential counter is SC$\wedge$(7), the second is SC$^{-1}\wedge$(8). The table 1 is a simple experiment for comparing of these two methods. Because the number of auxiliary variables depends on the number of variables to be true, the number of clauses of SC$\wedge$(7) is increasing in accordance with k and the number of clauses of SC$^{-1}\wedge$(8) is decreasing in accordance whit k. Hence CS(Compositional method of Sequential counters) is the third method which uses SC$\wedge$(7) when $k < \frac{n}{2}$ and SC$^{-1}\wedge$(8) when $\frac{n}{2} \leq k$.

Table 1. Comparison of SC $\wedge$ (7) and SC^{-1} $\wedge$ (8)

n/k	SC$^{-1}\wedge$(8)			SC$\wedge$(7)		
	#var.	#Clauses	Time	#var.	#Clauses	Time
100/20	2200	6017	0.031	8020	19099	0.062
100/50	5127	12725	0.062	5050	11224	0.031
100/80	8053	17630	0.093	2080	4249	0.031

Finally, in the point a view of negative representation, if the ADC adopt to SC^{M}, the formula (10) is generated, and it has $2(n-m)$ clauses.

$$\bigwedge_{i=1}^{n-m} e_i \Rightarrow s_{i+m-1,m} \quad \wedge \quad \bigwedge_{i=m+1}^{n} e_i \Rightarrow \neg s_{i-m,1} \qquad (10)$$

4 Experiments

In this section, through experiments we compare with each encoding method. We deal with solving puzzles such as figure 2, Japanese puzzle which is form of logic drawing: the puzzler gradually makes a drawing on grid, by means of logical reasoning.

The puzzler is provided with information about the horizontal and vertical arrangement of the black pixels along every line. For each line, the description indicates the sizes of the segments of adjacent black pixels, in the order in which they appear on the line. To consider Japanese puzzle as a SAT problem, firstly we allocated each Boolean variable into a cell as figure 2, and then for each line, CNF formula is encoded by ADC, finally for an overlapped cell, like the right side of figure 3, we introduce additional clauses to avoid misuse of variables.

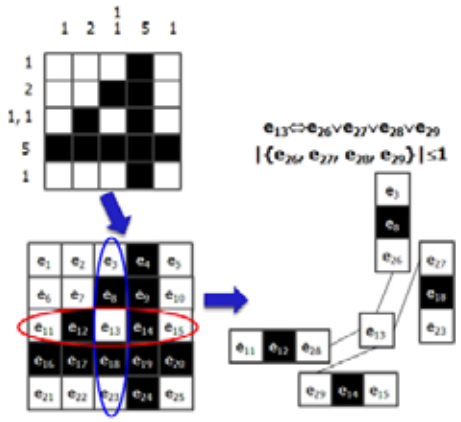


Figure 3. Encoding puzzle into CNF

Our experimental environment is follow: ubunto linux, 1.86GHz CPU, and 2G RAM. And we use Minisat solver[4]. We experiment different four methods for several Japanese puzzles. The label symbol "-" means to fail to generate CNF formula.

In the case of convex style, the number of clauses of TA is smaller than $SC^{M}\wedge(10)$, because of the shape of the data whose lines are consecutive and the method of $SC^{M}\wedge(10)$ generates the most clauses because it use equivalence operator in the sequential counter and the sequential counter is very sensitive to the number of variables assigned with true. However, the CS method makes the smallest clauses, and the reason is that the method computes which method generates the smallest clauses between $SC^{-1}\wedge(8)$ and $SC\wedge(7)$.

In the case of the random data, the $SC^{M}\wedge(10)$ method generates the smallest clauses and the method is the fastest. In the case of Japanese puzzles from 10×10 to 100×100, its results are similar to random data, but these data less complex than random style and more complex than convex style. Regularly the CS method shows good performance, but because it uses naïve ADC, so about bigger problems the TA method will be better than CS.

Table 2. The number of clauses

Problem	TA	$SA^{M}\wedge(10)$	CS	$SC\wedge(7)$
c10	3500	3576	1576	2768
c20	23740	30156	11340	20448
c30	73632	103736	36172	67044
c40	166000	248316	84136	156564
c50	312932	487896	161068	303004
c60	526404	846476	276360	520324
c80	1204400	2016636	645956	1223680
c100	2296164	3950796	1251168	2378872
c120	3897648	6840956	2149744	4097608
r10	3537	2473	2211	2596
r15	12627	8613	8636	9129
r20	30065	19688	20024	20849
r25	66845	41932	43129	44266
r30	120641	72755	76027	76997
r35	200403	118452	124257	125634
r40	316406	181614	191192	192474
r45	480998	269496	283107	283739
r50	689632	377894	399634	400152
10x10	4251	3060	2616	3074
18x10	5343	6132	2269	4460
15x20	23399	18485	12613	15999
20x20	37453	26206	23159	25253
25x25	66835	52076	42844	46570
40x40	345895	200919	185750	190410
40x79	1030041	575821	503588	515940
50x100	1752110	1142766	916629	1007446
100x100	5217099	2932747	2372118	2521863

Table 3. The number of variables

Problem	TA	$SA^{M}\wedge(10)$	CS	$SC\wedge(7)$
c10	780	1000	676	1172
c20	3920	8000	4684	8656
c30	9780	27000	15044	28464
c40	18880	64000	34728	66584
c50	31100	125000	66784	129024
c60	46320	216000	114092	221756
c80	88320	512000	266768	522152
c100	144400	1000000	516920	1015944
c120	214080	1728000	888324	1750880

r10	861	633	759	911
r15	2777	1999	2702	2885
r20	6117	4243	5852	6104
r25	12449	8299	11549	11953
r30	21017	13683	19505	19837
r35	33263	20984	30174	30696
r40	49339	30528	44397	44800
r45	70503	43329	62797	63006
r50	97218	57107	84173	84258
10x10	994	763	858	1085
18x10	1281	1721	844	1952
15x20	4023	4284	3916	5294
20x20	6777	5733	6846	7599
25x25	11069	11677	13005	14378
40x40	42917	37369	45547	47364
40x79	90420	99630	114222	118500
50x100	175199	202626	203305	240511
100x100	343959	511821	530367	581625

Table 4. Solving time

Problem	TA	SA$^{M}\wedge$(10)	CS	SC$\wedge$(7)
c10	0.003	0.003	0.000	0.003
c20	0.015	0.015	0.008	0.012
c30	0.045	0.059	0.021	0.039
c40	0.101	0.172	0.092	0.089
c50	0.195	0.404	0.096	0.167
c60	0.321	0.832	0.171	0.825
c80	0.715	5.115	0.420	9.254
c100	1.424	13.726	1.031	4.726
c120	2.441	31.787	1.733	2.679
r10	0.004	0.004	0.000	0.004
r15	0.016	0.012	0.012	0.012
r20	0.032	0.056	0.068	0.076
r25	0.140	0.176	0.172	0.192
r30	0.496	0.816	0.904	1.328
r35	10.668	2.188	4.300	1.868
r40	165.880	165.090	93.749	159.080
r45	40.234	25.985	11.244	11.536
r50	81.753	32.382	46.498	71.364
10x10	0.004	0.004	0.000	0.004
18x10	0.004	0.004	0.000	0.008
15x20	0.028	0.028	0.036	0.032
20x20	0.032	0.028	0.040	0.040
25x25	0.084	0.128	0.144	0.184
40x40	0.652	0.540	2.256	4.468
40x79	2.224	1.964	3.648	4.823
50x100	4.144	4.084	12.904	15.341
100x100	6.472	7.896	20.197	24.046

5 Conclusions

In this paper we explained several CNF encodings for BCC and proposed the concept of ADC. We could basically represent the ADC with two ways: positive and negative representation. And then to fine more efficient method we adopted methods using a binary tree and a sequential counter, and to apply ADC to those encodings for BCC, we modified the sequential counter with equivalence operator and newly design the inverse form of the sequential counter. Also we split the method using a binary tree into two parts: at least l part and at most m part.

Through experimental results, we confirmed the necessity of ADC to translate problems into CNF formula in the point a view of efficiency. The experiments were performed by three types of data: convex style, random style, and Japanese puzzle. Although TA, SC$^{M}\wedge$(10), and CS among methods showed good results similarly, we would predict, because of the size of generated clauses, that the TA method will be better than SC$^{M}\wedge$(10) and CS in cases of big problems.

References

[1] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem Proving". Communications of the ACM 5 (7): 394–397 , 1962.

[2] Marques-Silva, J. P., and Sakallah, K. A., "GRASP: A Search Algorithm for Propositional Satisfiability," IEEE Transactions on Computers, vol. 48, 506-521, 1999.

[3] M.W. Moskewicz, C. Madigan, Y. Zhao, L. Zhang and S. Malik, "Chaff: Engineering an Efficient SAT Solver," In Proceedings of Design Automation Conference, 2001.

[4] http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/

[5] H. Kautz and B. Selman. Planning as satisfiability. In Proceedings of the ECAI'92, pages 359–363. John Wiley & Sons, Inc., 1992.

[6] A. Biere, A. Cimatti, E. Clarke, Ofer Strichman, and Y. Zhu, "Bounded Model Checking", Vol. 58 of Advances in Computers, 2003.

[7] http://alloy.mit.edu/

[8] G. Kwon and H. Jain, Optimized CNF Encoding for Sudoku Puzzles, in The Proceedings of LPAR06, 2006.

[9] O.Bailleux and Y. Boufkhad, "Efficient CNF encoding of Boolean cardinality constraints," in Proceedings of the CP 2003, vol. 2833, LNCS, 2003.

[10] C. Sinz, "Towards an optimal CNF encoding of Boolean cardinality constraints," In Proceedings of the CP 005, vol. 3709, LNCS, 2005.

[11] K. J. Batenburg, An evolutionary algorithm for discrete tomography, Discrete Applied Mathematics, 2004.

[12] K. J. Batenburg and W. A. Kosters, "A discrete tomography approach to Japanese puzzles," in Proceedings of the Belgian-Dutch Conf. Artificial Intelligence, 2004.

[13] http://en.wikipedia.org/wiki/Nonogram

Bridging the semantic gap between process documentation and process execution

Gregor Scheithauer*, Guido Wirtz
Distributed and Mobile Systems Group, University of Bamberg
Feldkirchenstraße 21, 96052 Bamberg, Germany
gregor.scheithauer@gmail.com, guido.wirtz@uni-bamberg.de
Candemir Toklu
Knowledge Management, Siemens Corporate Research, USA
candemir.toklu@siemens.com

Abstract

Process documentation and process execution are part of companies business information systems which describe the way how companies process information. Between these two levels of abstraction exists a gap that has been identified as the main drawback for business process implementation. This paper proposes a holistic methodology using BPM on top of SOA to overcome this gap by providing an architecture that bridges these different levels of abstraction by transition strategies, and a life cycle to support transitions.

Keywords: BPM, SOA, Process Modeling, BPMN, BPEL

1. Introduction

Process documentation and process execution are part of companies business information systems which describe the way how companies process information [6]. Two levels are differentiated within business information systems; The first level compromises business tasks which are associated with information relationships, whereas the second level is the task bearer level. The latter consists of a set of task bearers which are associated by communication systems. Between both levels exists an allocation relationship which represents the mapping of tasks and task bearers. This classification supports the understanding of the difference between process documentation and process execution. Process documentation is part of the task level. It represents the sequence of tasks which need to be performed in order to run a business. Also, it comprises the substantiation of business goals and strategies, and is carried out by people with business domain knowledge. The methodology used for process documentation is a visual depiction of processes [25]. The degree of formalization is either non-formal or semi-formal. The purpose of it is threefold: to reduce business process complexity, to ease communication between business partners, and to communicate business logic for the task bearer level. Process execution is associated with the task bearer level and represents software applications. The purpose of it is to support automated business processes. The solution is a formalized software code.

*The first author is funded by means of the German Federal Ministry of Economy and Technology under the promotional reference 01MQ07012.

Mapping business tasks to software applications comprises several issues which need to be overcome in order to successfully map business requirements to software applications. The most prominent one, i.e., the loss of information during the mapping process [7, 19, 20, 23, 25], often results in a false translation from process models to implementation plans. The entirety of problems that occur in this context are often referred to as the *semantic gap*.

The aim of this paper is to offer an analysis of the semantic gap, and to examine the mapping between tasks and task bearers. Additionally, it addresses the issues described above by means of a holistic methodology, and offers a solution by narrowing the semantic gap.

Section two depicts work that is related to the holistic methodology. Section three presents a more detailed analysis of the semantic gap. Section four offers Business Process Management (BPM) [22] on top of Service-oriented Architectures (SOA) [25] as a holistic methodology to bridge the semantic gap. Section five summarizes the findings and presents future work to be done.

2. Related Work

The works of Decker, and Dehnert and van der Aalst also discuss interesting level concepts. Decker [4] differentiates between a business layer and an execution layer, and presents four strategies to map these. His work defines a process support layer, and patterns to support the mapping between the business layer and the execution layer.

Dehnert and van der Aalst [5] propose an approach to bridge the gap between business processes (EPC notation) and workflow specifications (Petri nets). Different levels of business process abstraction are addressed with different modeling techniques. The transition between different levels is implemented in a set of rules, which does not limit the designing facilities of the EPC notation.

Hofmeister's and Wirtz's [8] work relates to service integration. They present a pattern taxonomy to standardize the design of coupling systems and thus, to ease integration. To do this, they introduce a refined approach of Business Process Integration Oriented Application Integration (BPIOAI), which is based on message-based integration on a more ab-

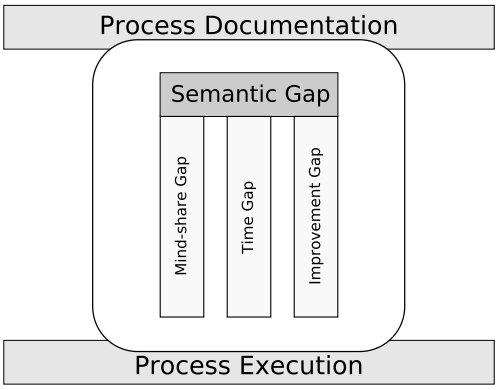


Figure 1. Different views on the semantic gap.

stract level. Regarding mapping and decomposition strategies, they propose an architectural framework to design composite applications on top of existing legacy applications. The framework includes service reuse, coupling, different layers of abstraction, and patterns.

Lastly, Model Driven Architecture is an approach by OMG to design software on the basis of formal models [11]. Its aim is to decrease the time to implement software applications. Platform independent formal models (PIM) are transformed into executable code.

3. Exploration of the Semantic Gap

The semantic gap is explained with the help of three different views to refer to the problems it implies; namely the mind share gap, the time gap, and the continuous improvement gap. Figure 1 summarizes these different views of the semantic gap between process documentation and process execution.

The *mind share gap* refers to the difficulty to translate business requirements directly into supporting applications. Business owners and software developers talk different languages, perceive problems and solutions differently, apply different methodologies, and use different approaches to communicate and understand business processes and requirements. These different contexts make it difficult to transform business requirements into executing software applications [20, 23].

The *time gap* refers to the time delay between the setting of business requirements and the development of the software applications. Software applications which support business logic are complex, involve a team of developers, need to be reliable, and need to be set up for a whole company. Hence, a great amount of time elapses between the moment of ordering the system and the moment of completion. In the worst case this could mean that business requirements change during software application development and consequently, the application becomes obsolete [7, 21].

The *continuous improvement gap* deals with the ability to refine business processes and the underlying supporting software applications. According to Woodley and Gagnon [25], simple steps in a business process do not change often. What does change often, are sequences and the integration of these into different business contexts. Right now, business requirements are hard-coded into software applications and cannot be changed easily. The association between original business processes, and applications supporting the process, is not formally captured and cannot be repeated or managed when processes need to be changed in order to adapt to new market needs. Moreover, as it is crucial for companies to accommodate their business processes according to their visions and business goals to match market requirements, it is not desirable anymore to hard-code business logic into one application but to store the business logic separately from implementation code. However, in a fast changing business environment, business requirements need to be adapted frequently [7, 21].

4. Holistic Methodology

This section presents a solution to bridge the *semantic gap*. The holistic methodology comprises (1) a level concept to clearly separate the task level and the task bearer level, (2) a life cycle to drive the whole methodology, and (3) transition strategies, which provide relations between levels. Every level serves as a classification for BPM goals, views on the semantic gap, the BPM life cycle, BPM stakeholders, design notations, execution languages, and technology. The level concept will be the basis for the transition strategies. Transition strategies are the bridge between process documentation and process execution.

The emerging BPM [22] technologies are providing an opportunity to manage the life cycle of the business processes for a company and therefore, effective deployment and execution of information technology [21] in order to support the bridging of the *semantic gap*. The BPM definitions from zur Muehlen and Ho [26], Gartner [10], van der Aalst et al. [22], and the BPM Standard Group[1] were considered for the derivation goals for BPM. Table 1 summarizes the findings about the essential BPM goals. Thus, BPM is a methodology to discover and document processes as well as to integrate software applications to support business processes in an automatic way, and it offers a life cycle to improve and adapt business processes according to changing business environments.

SOA supports BPM by offering a platform that supports business processes and integrates heterogeneous business applications and business partners. Business processes consume and leverage such SOA services, tying them together to solve business challenges [25]. BPM on top of SOA as a methodology provides a solution to narrow the semantic gap by solving the mapping problems and thus, to diminish the consequences. The independence between business processes and services helps to separate the business model and the technical implementation. The velocity of the implementation matches the speed of the quickly changing business requirements. Processes need to be independent from a specific platform. Otherwise, the logic is hard-coded into software applications and thus, expensive to change. Woodley and Gagnon [25] claim

[1] BPM SG, http://www.bpmstandardsgroup.org/resources.asp

Table 1. BPM definition comparison.

	zur Muehlen and Ho	Gartner Group	BPM SG	van der Aalst et al.
Business driven	X	X	X	
Flexibility	X	X		X
Integration	X			
Improvement		X	X	X
Automation		X	X	X

that both BPM and SOA are IT concepts. This is only partly true. IT is only one part of the BPM methodology since so many more aspects need to be considered, such as management issues, business strategies, and people. However, SOA serves as an integration platform for business services that are loosely coupled and easy reused [9, 12, 15, 20].

4.1. The Level Concept

This subsection introduces a three level concept: the business level, which represents business process documentation and improvement, and the integration level, which represents a SOA implementation. It orchestrates services that support business processes [15]. The third level is the execution level that represents business applications.

The *business level* addresses the process documentation. Business processes, business tasks, business objects, and business partners are documented with the aid of business process diagrams designed with process notations such as Business Process Modeling Notation (BPMN) [24] and Event-driven Process Chains (EPC) [17]. Business processes are fundamentally very abstract. They consist of a flow of business tasks connected by a control flow. Business objects refer to information that is processed by business tasks. The interaction between two companies within a business process describes business partners. Business logic is refined by process decomposition. This makes it possible for business analysts to describe the requirements for supporting software applications in detail and in a language that is understood by business analysts. Furthermore, the flexibility of process design notations allows to express business logic without limitations. This narrows the mind share gap. Lastly, ongoing improvement allows the continuous adaption of business processes to a changing business environment, which leads to a narrowing of the improvement gap. Regarding the life cycle management process design, process monitoring, and process improvement occur on this level by business analysts, process owners, and business owners. This level is neither technical nor platform specific. The flexibility to change business processes is very high.

The *integration level* serves two purposes: firstly, it is the target platform for formal and executable processes; secondly, it provides the means for a common uniform representation for services. Business Process Execution Language (BPEL) [1] allows the implementation of complex business processes on the basis of existing web services. The BPEL code represents the sequence flow of business processes registered on the business level. The service functionality is described with a specification such as the Web Service Description Language (WSDL) [3]. This specification also includes message type definitions which are processed by the service. A service represents an atomic business function. Otherwise, too much interaction is needed between services and partners. An integration-oriented architecture allows to integrate heterogeneous business applications onto a homogeneous level, which makes it easy to access and combine business applications to support business processes completely. Furthermore, the decoupling of services eases the change of business processes. In that case business logic may be changed on the business level. This change does not require an adaptation of a hard-wired business application. In fact, it is adequate to rearrange the supporting service using tools without the need to change application code, to recompile the code, and to redeploy the code with all the difficulties involved in application change management. This fact addresses the time gap and the continuous improvement gap. The integration level comprises the life cycle steps process configuration, service integration, and service development. The system analyst is responsible for process configuration, and service integration, whereas the software developer is responsible for developing the services. This level encounters middleware solutions, such as EAI, process configuration tools, and tools to deploy configured processes. The integration level is the soft glue between flexible business processes and hard-coded applications. Though the integration level is technical, it is still not platform specific. It is possible to take process configurations and deploy them in a different technical environment that supports open standards.

The *execution level* addresses business applications, legacy code applications, process execution engines, and other middleware. It forms the technical basis for automated business processes. Many business functions are hidden within business applications, which makes it difficult to use or even reuse them in loosely coupled business processes. The execution level comprises the life cycle steps service development, process deployment, and process execution. The software developer is responsible for service development and is involved in the process deployment step. This level encounters development and middleware tools. Thus, this level is technical and platform specific.

4.2. Life Cycle

Using the level concept, a life cycle to transform business process diagrams into executable processes will be introduced. Table 2 summarizes the findings of the life cycle concepts of Smith and Fingar [21], Netjes et al. [14], and the BPM Group.

Table 2. BPM life cycle comparison.

	Smith and Fingar	Netjes et al.	BPMG
Process Discovery	X	X	X
Process Design	X	X	X
Process Configuration		X	
Service Integration			X
Service Development			X
Process Deployment	X		
Process Execution	X	X	
Process Monitoring	X	X	X
Process Improvement	X		X

A comparison of the three life cycle concepts, resulted in a nine step life cycle for managing business processes.

Process discovery refers to the detection of business goals and strategies in order to conduct a business. A way to formalize goals and strategies are either ontologies or a business model [16].

Process design refers to the transformation of goals and strategies into a process diagram on the business level. In order to perform this step, two intermediate steps are required: first, business analysts will need a business process design notation. In this context, this might either be the BPMN or the EPC notation. Subsequent, business analysts will use their experience, design paradigms, and the purpose of the documentation for transforming the informal goals, strategies, and rules into a process documentation. The design paradigm is process-oriented. Business analysts decompose tasks and objects to substantiate business processes. Second, they need to simulate the process in the diagram to verify the semantical behavior of the process.

Process configuration refers to the transformation of process documentation into a platform independent process configuration. It implies the mapping strategy on the integration level. It is intended to transfer the process documentation as complete as possible, not to change any business logic, and to reuse existing implementations. To achieve this step, system analysts need to perform four steps: First, the process documentation is transformed into a technical representation. At this point, a switch in notation is not necessary. The configuration tool offers additional constructs to enrich the process documentation with technical details. Hence, the flow of business tasks (service composition) can be derived from the business process diagram. Second, web services are mapped to atomic business tasks. The integration is done by using middleware technology, such as webservices. Third, message descriptions need to be applied to web services. They also need to be maintained and integrated by the system analyst. Fourth, system analysts need to validate the work and build the process configuration. In case no service for an atomic business task is available, two possibilities are conceivable: consultation of a software developer, who develops an application function to address the business taskss requirements, or to integrate a service from a service provider.

Service integration follows either the development step, or the configuration step. It supports the integration strategy. Integration is twofold. Both, developed services, and services from business partners need to be integrated before they can be used for process configurations.

The development of an application function that is wrapped as a service and fulfills the requirement of an atomic business task is called *Service development*. The objective is to develop a service which is not bound to a specific business process resulting in a self contained service which is easy to reuse. Since business analysts break business processes down into loosely coupled business tasks and decompose business tasks (treated as business processes) recursively into atomic business tasks, the whole complexity and semantic is broken down into seizable and easy to communicate requirements, which are understood by software developers. Software developers grasp the requirements and build an application function using programming languages such as Java, or .NET and wrap this function as a reusable service.

Process deployment refers to the final transformation of the process configuration onto a platform. It supports the transformation strategy. However, the deployment should not have an influence on other running processes on that platform. The executable process configuration needs to be transferred to the execution engine. The configuration of other middleware technologies, which are involved in the process, as well as security policies, is optional.

Process execution alludes to the state where configured business processes are executed to support business processes. The procedure to execute a process differs in terms of how processes are triggered and whether the service is known in the first hand. First, process consumers need to know about the service. Second, processes need to be triggered. Process might be triggered by an event, other services or by human interaction through a process portal. Third, running processes are referred to as process instances. Process instances save the state of a process, as well as message and variable values.

The tracking of the process performance and to display it in a human understandable format is referred to as *Process monitoring*. The formal goals are to highlight bottlenecks and offer suggestions for action in order to improve process performances. Often, a dashboard is used as a paradigm. Business owners should be in the position to view the performance of process instances on a consolidated level. It should be al-

lowed to break the level down to a single instance which may be a source of failure or a bottleneck. Business owners are then allowed to take action, e.g. improve the process.

Process improvement points to the adaption of processes due to a changing situation or environment. Objectives are to improve execution time, execution cost (total cost of ownership), and process efficiency. Business analysts and business owners gather new information and feedback from process monitoring, process users and business owners. They change the process diagram according to the new requirements. The new process diagram needs to be validated. Moreover, depending processes need to be checked whether they are affected by the current change of the original process. Adapted processes are configured and deployed for execution.

4.3. Transition Strategies

Transition strategies address the transition between the business level and the integration level, and the integration level and the execution level. The mapping between the execution level and the integration level will be carried out once for every service on the basis of standards. The mapping between the business level and the integration level occurs every time a new business process is introduced or a business process is adapted, and thus, new configured and deployed.

The *decomposition strategy* starts with documented business processes and business objects using a process notation, such as BPMN, according to business strategies, goals, and rules, provided by business owners and process owners. These documentation may serve as a communication basis between business owners and business analysts. However, to communicate business logic to process participants on a more operational level, it is necessary to concretize the business logic. This is done by business analysts in connection with business owners. They *decompose* abstract business tasks. This is possible if business tasks are treated as processes themselves. The process notations BPMN and EPC support the decomposition strategy. Decomposition is an iterative procedure. In case a business task cannot be decomposed anymore without loosing business relevance, the decomposition stops. Tasks which cannot be decomposed anymore are referred to as atomic business tasks. Though decomposing business tasks reduces the complexity of business logic, the reduction in complexity has its limits, if business language is enriched with too many details. Furthermore, this hierarchical adjustment of business tasks allows business analysts to change business processes easily. The decomposition strategy narrows the mind share gap and the improvement gap.

The *mapping strategy* urges business analysts to create self-contained atomic business tasks for three reasons. Self contained business tasks improve the reusability of these business tasks, since they are applicable in more than only one process. Furthermore, self-contained business tasks are *mapped* more easily to stateless web services [25]. The semantics of self-contained business tasks is communicated to system analysts more easily, since no context knowledge is necessary. Thus, atomic business tasks are the ones to be mapped to services on the integration level, and atomic business objects are the ones to be mapped to message types. In conclusion, services must be available which correspond to atomic business tasks. According to Natis [13], the reason for coarse-grained services is that the message interaction between services does not need to be chatty. System analysts take over at this point. They are responsible to map atomic business tasks to business services. They need to search for an appropriate service, which might be a difficult task, in case the service repository is huge, and in case the semantic of the atomic business task is not well understood by the system analyst. In both cases, system analysts may consult business analysts for support, or system analysts are supported by clever tools. The mapping strategy addresses the time gap and the mind share gap.

The integration strategy is carried out by system analysts. System analysts are responsible for the service repository. They integrate services and message types from within the company and from external service providers. Thus, it is possible to use powerful software applications, in a standardized fashion. Business processes do not stop at application borders, they cross application functionality as well as department responsibilities. However, it is not intended to *integrate* services on the basis of availability. Since BPM should be business-driven, requirements for atomic business tasks and atomic business objects should be the trigger to develop or integrate a service. Thus, system analysts in connection with business analysts define requirements for services which need either to be developed by software developers on the basis of existing software applications, or to be searched in service repositories. The integration strategy supports the integration goal of the BPM methodology, and addresses the time gap.

The *transformation strategy* refers to the procedure of transferring the atomic business task orchestration [18] into executable code. Business analysts decompose abstract business processes into an orchestration of atomic business tasks. System analysts map these atomic business tasks to corresponding business services. The configured process is now ready to be transformed into executable code. All information required is provided by the mapping of atomic business tasks to web services. Executable code refers to an execution language which may be executed by execution engines, like, e.g., BPML [21], BPEL [1], or a vendor specific execution language. The transformation procedure is an automatic step, thus it is easy executed and does not limit the improvement of business processes. Transformation supports the automation and improvement goal, and addresses the time gap.

5. Conclusion and Future Work

The aim of this paper was to bridge the semantic gap between process documentation and process execution. Problems originating from the semantic gap were identified and classified into the three different views. BPM on top of SOA was in-

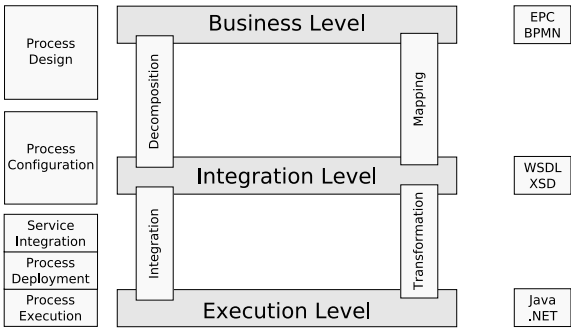


Figure 2. Architecture: Levels & Strategies.

troduced as a platform for a solution. The methodology comprises three levels, four transition strategies, and a life cycle. Figure 2 illustrates how the four transition strategies link levels, as well as how the life cycle corresponds to the levels and the transition strategies. In conclusion, the combination of the presented holistic methodology is capable to bridge the semantic gap.

Future work needs to address the strategic level and the transition between the strategic level and the business level. The strategic level is on top of the business level and covers business strategies and business goals, which are handled by business owners and process owners. Furthermore, existing process design notations need to be improved regarding the semantic of elements, and the methodologies how to use the notations to express processes. The EU-aided IP-Super project[2] addresses this question. Standards must be available to easily exchange process diagrams between stakeholders and tools. Process execution languages need to include more concepts to match business requirements. Patterns and standards for the transformation of process design notations into process execution languages must be improved. Moreover, existing tools and technologies must collaborate better to match requirements for BPMS. Finally, business-to-business integration must be raised to another level. Approaches, and tools must come available for the upcoming concept of service ecosystems, which Barros and Dumar depict in [2].

References

[1] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, and C. K. Liu. Specification: Business Process Execution Language for Web Services vers. 2.0. Tech. rep. 2.0, OASIS, Jan. 2007.

[2] A. P. Barros and M. Dumas. The rise of web service ecosystems. *IT Professional*, 8(5):31–37, 2006.

[3] D. Booth and C. K. Liu. Specification: Web Services Description Language (WSDL) Version 2.0 Part 0: Primer. Technical report, W3C, March 2006.

[4] G. Decker. Bridging the Gap between Business Processes and existing IT Functionality. In *Proc. 1th Int. WS on Design of Service-Oriented Applications (WDSOA'05)*,2005.

[5] J. Dehnert and W. Aalst. Bridging the Gap between Business Models and Workflow Specifications. *International Journal of Cooperative Information Systems*, 13(3):289–332, 2004.

[2]IP-Super, http://www.ip-super.org/, last accessed 2008-02-29

[6] O. Ferstl and E. Sinz. Modeling of Business Systems Using SOM. *Handbook on Architectures of Information Systems*, 1:347–368, 2005.

[7] M. Hammer. Reengineering work: Don't automate, obliterate. Technical report, Harvard Business Review, July 1990.

[8] H. Hofmeister and G. Wirtz. A multi-layered framework for pattern-aided composite application design. In *Proc. of the 11th World Multi-Conference on Systemics, Cybernetics and Informatics*, July 2007.

[9] F. Leymann, D. Roller, and M.-T. Schmidt. Web Services and Business Process Management. *IBM Systems Journal*, 41:198–211, 2002.

[10] M. J. Melenovsky, J. Sinor, J. B. Hill, and D. W. McCoy. Business Process Management: Preparing for the process-managed organization, June 2005.

[11] J. Miller and J. Mukerji. *MDA Guide Version 1.0.1*. Object Management Group, Framingham, MA, June 2003.

[12] Y. Natis. Service-Oriented Architecture Scenario. Gartner Research, ID Number: AV-19-6751, April 2003.

[13] Y. V. Natis. Service-Oriented Architecture (SOA) Ushers in the Next Era in Business Software Engineering. *Business Integration Journal*, May 2004:23–25, May 2004.

[14] M. Netjes, H. Reijers, and W. v. d. Aalst. Supporting the BPM life-cycle with Filenet. In 18th Int. Conf. on Advanced Information Systems Engineering (CAiSE'06), ed., *Proc. of the EMMSAD Workshop*. Namur University Press, 2006.

[15] J. Noel. BPM and SOA: Better Together. IBM Website, White Paper, 2005.

[16] A. Osterwalder, C. Parent, and Y. Pigneur, editors. *Setting up an Ontology of Business Models*. Faculty of CS and Information Technology, Riga Techn. Univ., Riga, Latvia, 2004.

[17] A.-W. Scheer and M. Nuettgens. Architecture and Reference Models for Business Process Management. *Lecture Notes in Computer Science*, 1806 / 2000:376–389, 2000.

[18] A. Schoenberger and G. Wirtz. Using Webservice Choreography and Orchestration Perspectives to Model and Evaluate B2B Interactions. In *Proc. of SERP 2006*, volume 2006, pages 1–7. CSREA Press 2006, June 2006.

[19] H. Smith. Business Process Management–the third wave: Business Process Modelling Language (BPML) and its Pi-calculus foundations. *Information & Software Technology*, 45(15):1065–1069, 2003.

[20] H. Smith. The Emergence of Business Process Management. *Information & Software Technology*, 45(15):1065–1069, December 2003.

[21] H. Smith and P. Fingar. *Business Process Management, the third wave*. Meghan-Kiffer Press, 1th edition, January 2003.

[22] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske. Business Process Management: A Survey. In W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, editors, *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 1–12. Springer, June 2003.

[23] L. Verner. BPM the Promise and the Challenge. *ACM Queue: Tomorrow's Computing Today*, 2(1):82–91, March 2004.

[24] S. A. White. Specification: Business Process Modeling Notation Specification, February 2006.

[25] T. Woodley and S. Gagnon. BPM and SOA: Synergies and Challenges. In A. H. H. Ngu, M. Kitsuregawa, E. J. Neuhold, J.-Y. Chung, and Q. Z. Sheng, eds., *WISE*, Vol. 3806 of *Lecture Notes in Computer Science*, pages 679–688. Springer, 2005.

[26] M. zur Muehlen and D. T.-Y. Ho. Risk Management in the BPM lifecycle. In C. Bussler and A. Haller, eds., *Business Process Management Workshops*, vol. 3812, pg. 454–466, 2005.

Performance challenges in migrating to SOA based healthcare systems

Suyog Gaidhani, Vijayananda Jagannatha
Philips Healthcare,
Bangalore, India
{suyog.gaidhani,vijayananda.j}@philips.com

Abstract

The increasing need to deliver healthcare in a pervasive and cost-effective manner has led to a steady rise in the prevalence of distributed hospital environments. This has led to the migration of existing healthcare systems to Service-Oriented Architecture (SOA) based environments. The benefits of SOA to provide for a loosely coupled system and ease of integration of existing applications are well known. However these benefits are accompanied with several challenges, especially those related to ensuring the existing levels of performance. This paper discusses our experiences in a SOA migration with an emphasis on the techniques that we have used to address performance related challenges.

1. Introduction

Traditionally, hospitals have operated in a single location mode with all possible medical specialty departments housed under a common roof. This leads to the problem of access to specialist radiologists and diagnostic equipment being restricted to only these large medical centers. Rapid strides in communication networks and distributed computing have led to many hospitals adopting a distributed network of geographically spread locations. Such a hospital network typically consists of a central location similar in capabilities to the single location hospital discussed earlier. The remote locations are scaled down units provided with diagnostic equipment along with basic viewing and archiving facilities. Expensive resources such as post-processing systems capable of performing operations like image segmentation or computer-aided detection are usually located only at the central location with the ability of making the results available at any remote location. Remote access to medical databases becomes essential in such scenarios.

Existing healthcare applications need to scale up to meet these changes. Given their complex and heterogeneous structure, developing large scale distributed healthcare systems with stringent performance and security requirements poses a daunting challenge. Service-Oriented Architecture (SOA) [7] is an enabler to design such systems. This is due to its inherent support of loosely coupled design and ability to deploy services based on existing applications or new ones in a platform agnostic way.

Consider a typical large hospital with centralized healthcare applications as shown in Figure 1.

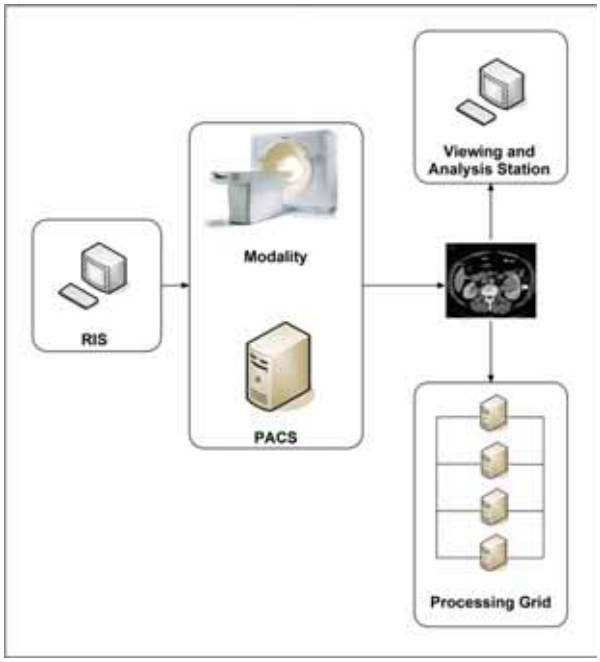


Figure 1. Existing monolithic healthcare enterprise

In such a system a patient diagnosis workflow will involve the following entities – a Radiology Information System (RIS) server, a Picture Archiving and Communication System (PACS) and a Viewing and Analysis Station. A patient with a referral for a scan walks in and completes the registration with the

relevant personal information and the required scan details. This information is stored in the RIS server and the scan scheduled on the required modality. After the scan is performed, the images are archived onto the PACS and the RIS updated with the scan results. Later, the images are retrieved for diagnosis and further recommendations by the concerned physician or specialist and viewed on the Viewing and Analysis Station. All the entities discussed are located in one single location.

If we model the existing monolithic healthcare system on a SOA based model (as shown in Figure 2), the RIS and the diagnostic equipment will continue to be located locally.

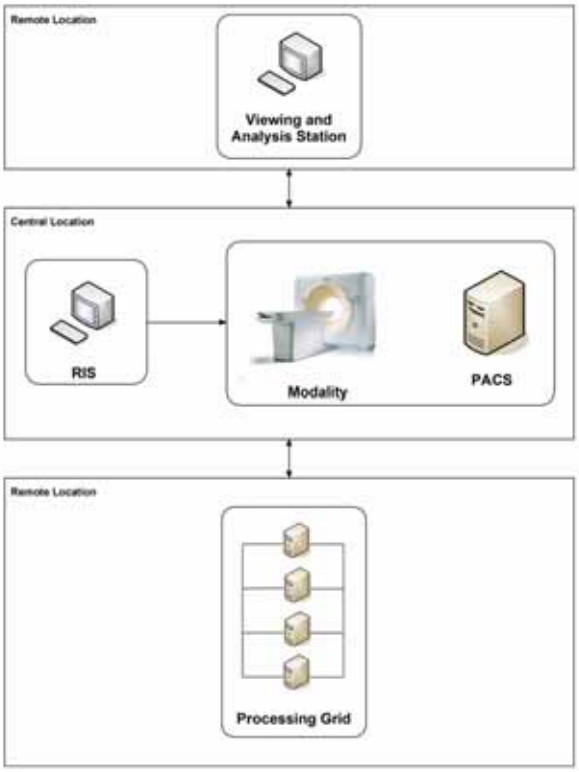


Figure 2. SOA based healthcare enterprise

This is to ensure that the patient specific systems should be as close to the patient as possible. Due to bandwidth considerations in the transfer of images, the PACS is also almost always located local to the diagnostic equipment. However, the Viewing and Analysis station is available at another location as is the post processing grid. As soon as the data is made available in the PACS, specific patient studies can be viewed and analyzed at any location by the concerned specialist. Relevant post processing which requires the usage of expensive hardware resources can also be performed remotely. All these systems will communicate via services based on standardized interfaces and data contracts, making the entire setup agnostic of the platforms, vendors and infrastructures.

2. Migration Strategy

Once the decision to migrate SOA has been made, the next step is the actual migration process. A typical SOA migration strategy is depicted in the Figure 3.

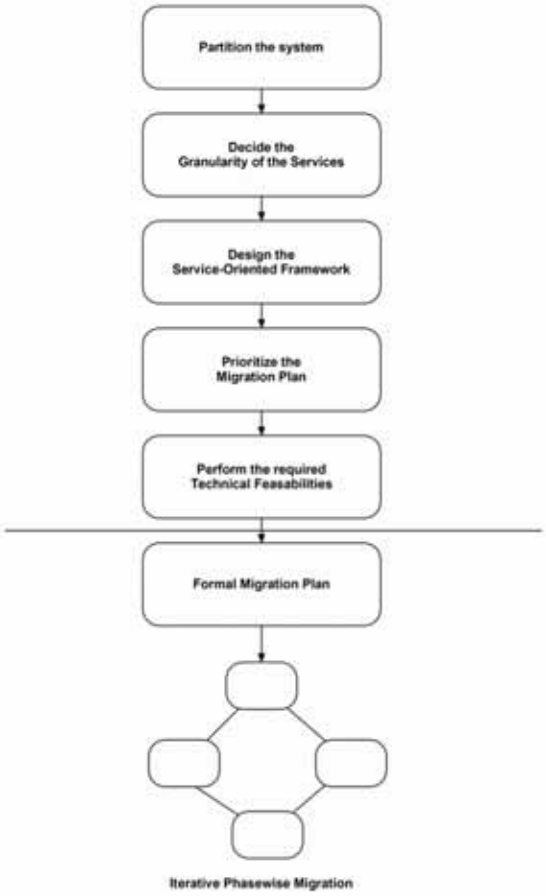


Figure 3. Migration strategy

The initial step in the migration process is to partition the existing system into a set of independently deployable services. The choice of the services may be driven both by business needs, the current architecture and future deployment considerations.

Figure 4 depicts the taxonomy of a typical healthcare software organized into functional layers. The lowermost layer is composed of application infrastructure components like Logging, Licensing,

Process and Thread management etc. These are services that are available across the entire application. Progressing further upwards are the healthcare infrastructure components like the DICOM (Digital Imaging and Communications in Medicine) Services, Imaging and Rendering Algorithms and Job Handlers which make use of the Application Infrastructure services. Further up is the business layer which provides core business functionalities. The granularity of the service and the composition of the system are decided based on the requirements and the deployment scenario. Some of these components can be exposed as individual service or may be aggregated together, for instance, one may look for a clustered approach to Job Handling where a single job (e.g. image reconstruction) can be broken into individual concurrent units and can be executed on multiple processing nodes while one may bundle the DICOM Service with the connectivity service in the Business Layer.

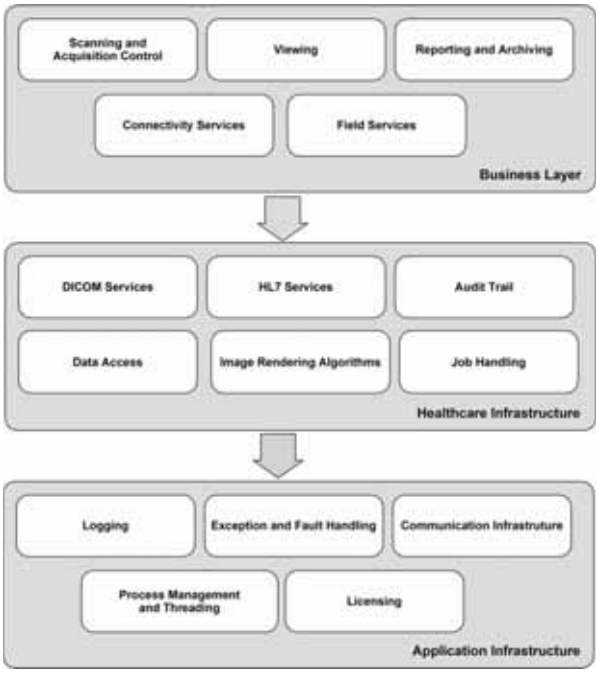


Figure 4. Taxonomy of healthcare software depicted as functional layers.

Service development encompasses a common set of activities both during the development and deployment phase. It is essential to abstract these boiler plate functionalities in a framework on top of which the services can be built. The framework provides support for elements like Service Discovery, Hosting, Logging, Exception Handling, Monitoring and Recovery etc. This technique,

- Avoids duplication of common functionality across services.
- Provides uniformity and a common feel for all services in the system.
- Enhances the productivity of the developers by enabling them to concentrate on the functional cases rather than plumbing the boiler plate code.
- Makes maintenance easier since the boiler plate code is confined to the framework instead of being scattered throughout the application.
- Decreases the Time-to-Market (TTM) by providing a better starting point for building applications.

The subsequent step in the migration process is technical feasibility analysis. This step examines the technology, tooling and framework support for carrying out the migration process. Chief use cases in the product may be prototyped to demonstrate proof of concept. It is also equally important to make sure that the new framework or technology does not in any way hamper the existing performance and scalability of the product. Additionally the cost of migration must not offset any perceived benefits of the migration.

Finally a phase wise migration plan is created taking into account the organization's business strategy, commitment to existing customers as well as the results of the technical feasibility phase and a clear road map is chartered out. The process involves interacting with the various stakeholders of the system in order to prioritize the features based on the business goals and resource availability.

The final step is the actual migration which is carried out in a phased manner. However, the process of migration has challenges related to various aspects of the system such as those related to security, deployment, versioning, reliability and performance. In the next section, we focus specifically on the performance related challenges and the techniques used to overcome them.

3. Overcoming performance challenges

Performance in a distributed environment is limited by a number of factors including the available network bandwidth, traffic, current load on the system etc. It is normal for a radiologist who reviews a certain number of images at a dedicated PACS reading room to expect the same kind of performance no matter where he/she is physically located with access to a medically certified display unit. For SOA implementations with large numbers of users, services, and traffic, maintaining the necessary performance levels presents

a substantial challenge. It is advisable to incorporate performance elements of the system into the design phase itself. The idea is to provide a reasonable performance if not the same kind of performance in a PACS reading room. Some of the techniques that we have used to gain performance are presented below.

3.1 Efficient Management of Resources

Certain resources in the system are expensive to create or scarce which accentuates the need for an efficient and a more robust technique for resource management. This can significantly improve the performance as well as the scalability of the system.

As an example, consider an enterprise PACS backed by a database like SQL server. At peak loads the system might experience an overwhelming number of connections. This significantly degrades the performance of the system. However, not all connections to the system might be currently used (a client can hold an idle connection). Figure 5 depicts the scenario in a typical componentized architecture where multiple clients load the data access component in-proc and establish individual connections with the database.

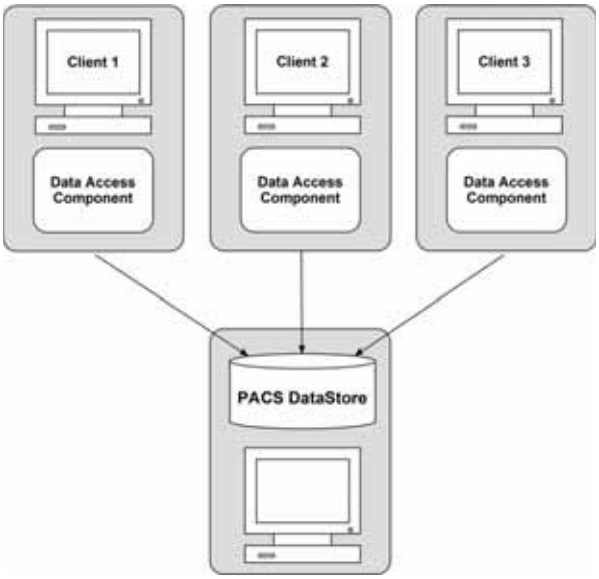


Figure 5. Non-pooled data access in a componentized architecture

SOA provides a solution to solving problems of this genre. Wrapping a light weight service oriented wrapper around the PACS data store lets the system to cap and pool the incoming connections. Connections can be efficiently reused without having to re-create them. This increases the performance as well as the scalability of the system. Figure 6 depicts the scenario in a typical SOA architecture where a dedicated data access service provides a single entry point to the physical data store and in turn throttles the number of incoming connections by reusing a set of pooled connections.

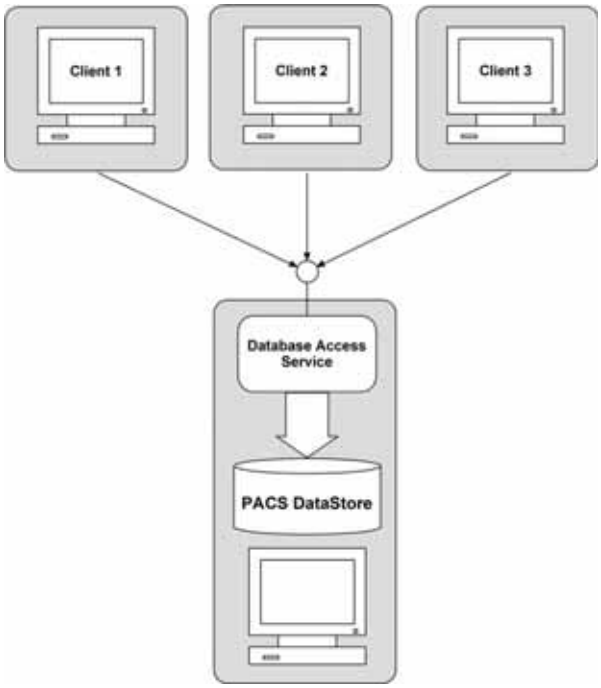


Figure 6. Connection pooled SOA data access

3.2 Optimized Image Transmission

Region of Interest or ROI essentially is a particular area(s) of the image of which radiologists are interested in to arrive at their diagnostic conclusions. Consider a case wherein the ROI of the generated image constitutes hardly 40% of the overall image. Representing the regions other than the ROI in high resolution simply increases the memory foot print of the viewer without adding any significant value to the user. Figure 7 represents a typical diagnostic image and its corresponding ROI.

Now, take the scenario of a radiologist accessing diagnostic images stored in a hospital PACS from a remote location. To start with, the radiologist can be supplied with images having only the ROI portions encoded in high resolution and the non-ROI portions encoded at a lower resolution. Only if the radiologist wants to access the complete image, will the entire image be delivered at its highest resolution. This technique conserves bandwidth, increases the

throughput and reduces the memory foot-print of the viewer.

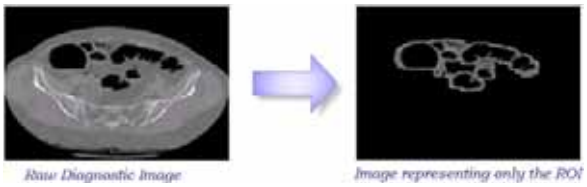


Figure 7. ROI based image encoding

Another alternative to conserve bandwidth and provide faster response times is to embed intelligence in the viewing terminals which will evaluate the way images need to be served to the radiologist. So instead of transmitting gigabytes of data at a time, small chunks of data are transmitted that would give just enough information for the radiologist to view at a time. Only when the radiologist tries to access more images or requests for the existing images at a higher resolution, will further data be fetched from the PACS and delivered to the radiologist. This technique, referred to as "Just-in-Time Delivery" or "On-the-Fly Compression", is increasingly making traditional needs of compressing the data itself redundant. [6]

3.3 On-Demand Data Transfer

Continuing with the use case of a radiologist at a remote terminal, in most cases he/she will want to request the studies of patients assigned to him/her. The meta-attributes of the patient (like name, sex etc) rather than the pixel data is what assists the physician to navigate to a study of interest. Only when the radiologist navigates to the detailed study is the pixel data actually needed. This underscores the need for On-Demand data transfer i.e. only the meta-attributes are fetched in the first shot and subsequent requests for diagnostic images fetches the image or the pixel data. Thus while migrating to SOA it is important to segregate these systems for metadata access and pixel data access.

For transfer of image data, SOA streaming can be employed where only the message headers are buffered without buffering the entire pay-load. This reduces the memory foot print of the application, the network traffic and also increases response times. Segregating the system this way lets it scale independently with different concurrency levels and different deployment considerations. For instance, the Image Streaming service is expected to be more resource intensive. Additionally, since the data is not buffered prior to transmission (streamed), it is essential to implement a custom reliability management technique here.

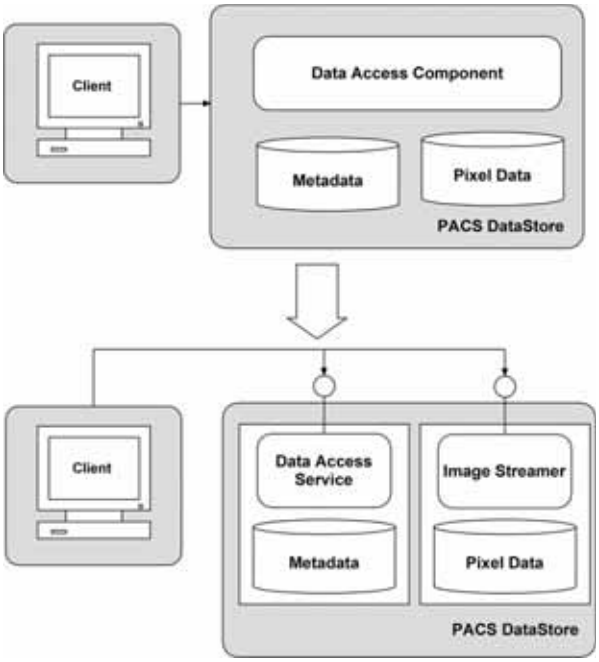


Figure 8. Segregating data access components in SOA

This Data Access Service should also be intelligent to eliminate redundant data. Take the example of a typical MR Scan which will involve about 600 images in a single study. All these images will have the same meta information until the series level. Hence it would make sense to send this kind of information only once instead of sending it every time with the images.

3.4 Choice of Messaging

In SOA, service interactions are typically characterized by accessing and invoking clearly defined service contracts. However, in some of our applications especially those related to notification of status messages, this style of method invocation is not a suitable option since inherently, these messages are asynchronous by nature and an entity sending this message is decoupled from the receiver of the message at the other end. Apart from this, there can be multiple entities receiving messages from a single entity. A traditional method invocation mechanism providing such an asynchronous one-to-many type of messaging would be inefficient especially with respect to performance. This is primarily due to the cost of setting up and executing the method calls for such a scenario

being non-trivial. The other option that was examined was of services polling for these messages from a queue or a repository. However, this was ruled out due to the increased network load that this polling would cause. In addition, most of the data would be unchanged leading to wasted calls. We finally opted for an open standards based messaging technology with a publisher-subscriber mechanism to provide a decentralized messaging infrastructure to meet our requirements. Using this mechanism, services interested in specific events subscribe to them and get notifications when there is an occurrence of those events. Our initial performance measurements lead to us believe that as the system scales up, the publisher-subscriber mechanism results in better performance as shown in Table 1.

Event Generation Rate (/min)	Event Receiver Count	Time taken to receive the event (msec)	
		Publisher/ Subscriber	Method Invocation
10	10	30	30
50	50	50	50
100	50	4800	6000

Table 1. Performance Measurements

4. Open Issues

Despite the fact that we have managed to meet the existing performance requirements in diagnostic remote viewing, it is still difficult to accurately predict the actual performance when deployed in the production scenario. With the deployment being at various levels scaling from a few nodes to hundreds, the task of measuring the performance is not easy as the performance depends on many dynamic parameters like concurrent users, bandwidth and service etc.

The other related open issue concerns the testing strategy. All along the strategy for the existing systems has revolved around test clients and test infrastructure keeping in mind the client-server or single box solutions. In theory, these should have been deployment agnostic and developed in conjunction with the interfaces exposed by the various components. However, in practice this has not been always true. Hence we are looking at several testing models where the current test cases can be executed in the new SOA enabled systems and accommodate the new ones as well.

5. Conclusions

SOA has emerged as a leading contender in moving existing healthcare delivery systems to a distributed environment. In our case, taking into account the diverse and large-scale nature of our existing healthcare applications and the need to overcome the challenges posed on the performance front, a phased approach to migration has been adopted. The remote diagnostic system was selected to be the first system for the transition. The initial results are encouraging as already highlighted in the earlier sections and we expect the other applications such as Remote Servicing, DICOM services among others to also move to a SOA based architecture in the long run. It is of our opinion that the general strategy of migration and the techniques used for handling the performance related challenges will prove to be useful guidelines when taking up migration in other healthcare applications where distributed computing is becoming a necessity.

6. References

[1] S.Van Assche, D. De Rycke, W. Philips, and I. Lemahieu, Exploiting interframe redundancies in the lossless compression of 3D medical images. Data Compression Conference, pp. 575, 2000.

[2] A.Vlaciu, S. Lungu, N. Crisan, and S.Persa, New compression techniques for storage and transmission of 2-D and 3-D medical images, Advanced Image and Video Communications and Storage Technologies, Amsterdam, Netherlands, vol. 2451, pp. 370-7, March 1995.

[3] Sarah A. Rajala, Majid Rabbani, Progressive ROI coding and diagnostic quality for medical image compression, Visual Communications and Image Processing Proc. SPIE, vol. 3309, pp. 674-685, 1998.

[4] Channabasavaiah, Holley, Tuggle, Migrating to a service-oriented architecture, IBM DeveloperWorks, 16 Dec 2003

[5] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns – Elements of Reusable Object Oriented Software, Addison-Wesley Pub Co., 1995

[6] Paul Chang, Enterprise Integration Strategies toward the Image-Enabled HER, HIMSS 2007, New Orleans, 2007.

[7] Thomas Erl, Service-Oriented Architecture - Concepts, Technology and Design, Prentice Hall, 2006.

[8] HIPAA. www.hipaa.org

[9] DICOM. www.medical.nema.org

Dynamically Optimize Process Execution Based on Process-Agent

Jian Dai[1, 2], Junchao Xiao[1], Qing Wang[1], Mingshu Li[1, 3], Huaizhang Li[1]

1 Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China
{daijian, xiaojunchao, wq, mingshu, hzli}@itechs.iscas.ac.cn
http://www.cnsqa.com
2 Graduate University of Chinese Academy of Sciences, Beijing 100039, China
3State Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences Beijing 100190 China

Abstract

Generally, one often uses his experience to optimize his action to achieve a certain goal. However, it is not true for the software process to help software engineers in making decision. Mainly there are two difficulties in dynamically optimizing software process execution; at first, many factors such as effort, cost, quality etc., are need to be considered and also the complex relationship among them, Secondly, there are different set of actions that should be taken with reference to the human resources' capabilities. So, in this paper we propose an approach based on Process-Agent to dynamically optimize the software process.

1. Introduction

Many aspects are involved in software process execution, such as cost, schedule, and quality. Generally speaking, Goal/Question/Metric (GQM) is used to support the quantitative evaluation of software processes and products [1, 2]. However, to implement the GQM, as [3] points out, the major problem is using such metrics in isolation, each indicator can just measure one aspect of the software process execution. However goal of a software project plan is often made up of different concerns. Project managers have to make tradeoffs among many aspects. To solve this problem, a method is proposed in [4] using the idea of separating concerns. By using this idea in software process diagnosis, the complicated and correlated aspects can be expressed clearly. However, due to the fact that the capability of individual is quite varying, we can not use one skilled coder's experience to a rookie. So the usage of this method is limited to fixed capability human resources and organizations.

Since human resources with varying capacities can use different experience to direct their action, we introduce the concept Process-Agent in this paper, defined at [5]. The Process-Agent represents a group of human resources having the same capability including skills, processes, etc. we can treat one process agent in the process execution as one component defined at the well-known Model-Based Diagnosis method [6-8]. Because each process agent concerns only a few activities in the whole process, we eastablished related diagnosis experience of it according to histoical executions of activities. The core concept used is state; we define it as a combination of the activity, measurement indicators, measured resluts and projects. So when the process agent perform the activity again in a new project, it can first measure its current state; and then comparing current state with the old ones to find the most similar state; finally according to the most similar one, it can predict what will happen next and provides some useful actions to the current project managers or current implementers to reach the next best one recorded in the historical data.

In sum, the approach we proposed uses a set of indicators measuring the historical data, distributes the measured results to each process agent, and updates the experience drawn from history through the current goal to figure out which one is better to be followed as an example and which are not so good states and should be avoided.

Rest of the paper is structured as follows. Section 2 is a brief introduction of related work. Section 3 presents the concepts used in this paper. Section 4 gives the details about approach as well as an example to demonstrate how to use this approach. Section 5 gives a discussion to make a conclusion and future prospects of our work.

2. Related Work

2.1 SoftPM and Process Agent

In [9, 10], Institute of Software Chinese Academy of Sciences (ISCAS) presented a solution for software process management and also implemented a toolkit: SoftPM. In [5, 11] Process-Agent was used to organize the process assets, and based on it, a process modeling method called an Organization Entity Capability Based Software Process Modeling (OEC-SPM) was presented to model standard processes. In this paper, we extend the process agent's experience library by adding states transformed from historical execution of process with the help of the software metrics defined in SoftPM. Our example process in this paper is a real process used in a CMMI4 company and its two executions are drawn from the SoftPM.

2.2 Problem Diagnosis

There are many problem diagnosis models and approaches used in the artificial intelligence field, which provides guidance to figure out problems in running process.

[12] is a traditional state based approach, and gives a new problem solver called STRIPS that attempts to find a sequence of operators in a space of world models to transform a given initial world model into a model in which a given goal formula can be proven to be true. When we try to use it in software process, it performs well if we treat the robots as no capability difference.

[13,14] give a simple framework and protocols for presenting plans, resources and goals of agents, where plans are defined as directed acyclic graphs of skills and special resources can be realized from given adequate initial resources, called goals. In this work, we extend this idea to software process using state and process agent to represent the historical executions of process, where an execution can be viewed as a directed acyclic graph of states linked by process agents.

the concept problem is defined in another way that is the state with low value evaluated by the value function. Here, problems may include exceptions, but we focus on optimizing process execution, so we don't distinguish them, just try to identify better one and problems to give clues for optimization. Our contribution in this paper also includes extending the oridinary diagnosis methods by predicting what will happen next and optimizing the process agent's action to avoid upcoming problems. Hence with the support of each process agent's optimization the process execution will be done in a better and smoother way.

3. Definitions

As metioned earlier, the execution of organization software process often considers many aspects. However, each measurement indicator often focuses on one aspect of execution, and we believe even if we only want to change one aspect of the execution, we should not only observe one aspect and take actions only on that aspect, because those aspects are highly correlated, and the relationships are often complicated. Here we establish process related experience of the process agent by using multiple indicators to measure the historical process execution. And when the process agent execute the same process, it can refer to the already eastablished experience to make a decision on what actions should be taken to reach the wanted state.

Here, we give some definitions used in this paper:

Definition 1. State

State is a triple S = (Set (activity, begin_end), Set (project), Ordered Set (Indicator, result)), where

(a) Set (activity, begin_end) is a set including activity and its status pairs. And each pair represents what the activity is and when the state is measured i.e. either at the beginning or at the end of the activity.

(b) Set (project) is a historical project set, representing from where the state has been measured.

(c) Ordered Set (Indicator, result) is an indicator and its measured result set, representing which indicators are used and what are the resultant states.

Definition 2. Value function

Value function is defined as: Result of $Indicator_1$* α + Result of $Indicator_2$* β +⋯+ Result of $Indicator_i$* δ , where Indicator i is selected to measure current goal,and α … δ is used to represent the weight of the measured result of specific indicator.

Definition 3. Edge

An edge is a quadruple E= (Set (process agent ids), value, state_from, state_to), where

(a) Set (process agent ids) contains process agents who lead to the change from one state to another.

(b) Value is calculated by the value function got from current project.

(c) State_from represents from where the edge begins.

(d) State_to represents to where the edge ends.

the whole concepts can be briefly viewed in Fig. 1.

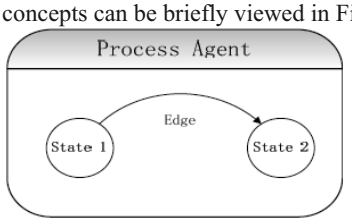


Fig. 1. Brief overview of the relationship among state, edge and process agent

4. Solution

In this section, we present overview of our approach; next we give the details about each step in our approach.

4.1 Approach Overview

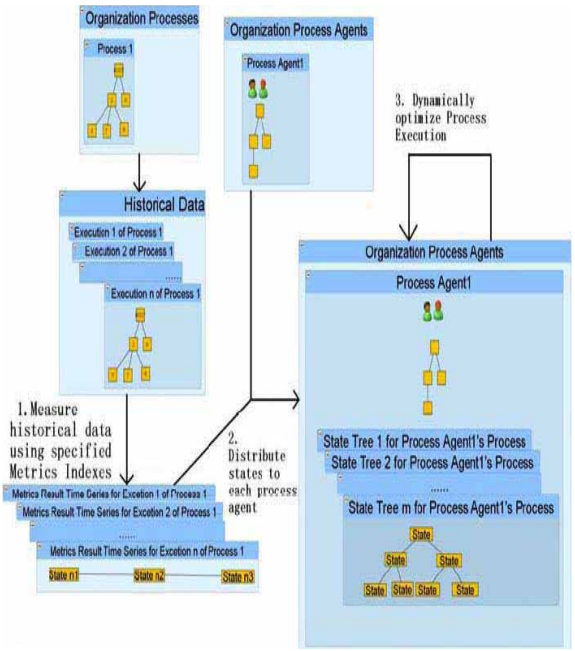


Fig.2. Approach overview

Fig. 2 shows the approach overview. Here, we assume that standard processes have been predefined in an organization and process agents have been established, which means we can get the relationship between historical tasks and activities in a specific process. Moreover, based on the human resources assignments in historical tasks, we can find the relationship between historical tasks and activities in a specific process segment of a specific process agent. So after many executions of a process, huge historical data can be accumulated during the process. Our approach mainly consists of three steps. First, we measure historical data using predefined metric indicators to get one state sequence for one historical execution. Then, combining with process knowledge in each process agent, we distribute states to each process agent to establish the state trees in order to represent its experience. Finally, after updating the value in the edges of states in each process agent by current project's value function, we can locate currently observed state obtained from measured results on the process agent's state trees, then forecast what has happened based on historical data and provide solutions to optimize current execution.

4.2 Measure historical data using specified Metrics Indexes

In this step, we measure process' historical execution data so that we can get one state sequence for each individual historical process execution. The procedure can be divided into three sub-steps, for the data related with each historical project: first, we get related processes by searching organization standard processes. Because it is hard to consider the similarity between historical state and current state without support of the same process, in this paper, we ignore those historical data that cannot tie with process; second, for each related process, measure the execution data at the beginning and end of each activity by a set of pre-defined metric indicators. Here, the set of indicators is used to reflect historical execution states, so the selection of the indicators should comply with the rule that the set of indicators should represent different dimensions to provide enough historical information as reference to current execution; third, after getting the measured results sequence, the state node can be built by combining historical project information and process information with each measured results sequence node.

4.3 Distribute states to each process agent

In this step, state sequences are distributed into each process agent. In order to decide which process agent the state belongs to, firstly, human resources are used as one condition. By observing the time when the state is measured, we can get related tasks undertaken and then can get the participants of the tasks. After that, to decide which activity in the process agent the state belongs to, we use the activity information in the task. After getting both human resources information and activity information, the state can be distributed to specific process agent under specific activity as a historical state. Since there may be already many states existing under the activity, we need to merge the new one with the old one. The merge rule is obvious: if all values in a new state are same with another existing state except the Set (project), then we can merge them by merge Set (project), otherwise the new state will be linked as similar to the old states belong to the same activity in the process agent.

4.4 Dynamically monitor and optimize Process Execution

In this step, we can make the prediction by locating current measured state on historical state trees built previously. Though we may not find the identical state on the trees, we can still give some advice based on most similar states. By comparing the most similar state's next states with current state, we can optimize the process agent's action towards a better direction. Here, we define better direction by introducing the value function, which is got by the weighted average of measured results calculated by indicators got from current running project goal using GQM. Once a new project comes, all the

weights on the edges are re-calculated, for historical states just express what happened, while which state is better is decided by the goal of current running project.

This step is divided into four sub-steps: first, we can draw indicators from all indicators which are relevant to the goal of the current project to form the value function according to the importance of different aspects; second, we update weight values for all edges from the results calculated by value function for each historical project; third, by comparing current measured state with historical states, we locate current result in state trees; fourth, we can provide optimizing advice by comparing the next best state with current measured state.

5. Conclusion

It is important to learn lessons from historical data, to direct current projects. It is quite common about an individual to use his experience to direct his current action. However, the key points to model this procedure in software process are: experiences from historical projects are hard to reuse, because they might be different in terms of goals, human resource's experiences are also hard to reuse, due to the varying capabilities. To solve these, we introduce the GQM idea to relate goals to specific indicators and the concept of process agent to organize historical data based on the capabilities of human resource. Basically, we propose an approach which treats historical data as only facts, organize them according to different human capacities, and use current project's goal to update the facts to help project managers dynamically optimize current project execution.

Acknowledgments. Supported by the National Natural Science Foundation of China under grant Nos. 60573082, 60473060,90718042 and the Hi-Tech Research and Development Program (863 Program) of China under grant No. 2006AA01Z185, 2007AA010303, as well as the National Basic Research Program (973 Program) of China under grant No. 2007CB310802.

We are also grateful to many teachers and students in the Laboratory for Internet Software Technologies. In particular, we wish to thank Professor Yongji Wang, Assistance Professor Juan Li, Qiusong Yang, M. Wasif Nisar, Jian Zhai, Dapeng Liu and Lizi Xie for their advice, support and encouragement.

References

[1] A. Fuggetta, L. Lavazza, S. Morasca, S. Cinti, G. Oldano, E. Orazi: Applying GQM in an industrial software factory, Vol. 7. ACM (1998)

[2] Van Latum, F., Van Solingen, R., Oivo, M., Hoisl, B., Rombach, D., Ruhe, G.: Adopting GQM based measurement in an industrial environment. Software, IEEE **15** (1998) 78-86

[3] Fenton, N.E., Neil, M.: Software metrics: successes, failures and new directions.Journal of Systems and Software 47 (1999) 149-157

[4] P.Tarr, H. Ossher, W. Harrison, S.M. Sutton, Jr.: N degrees of separation:multi-dimensional separation of concerns. Software Engineering, 1999.Proceedings of the 1999 International Conference on (1999) 107-119

[5] Q. Wang, J. Xiao, M. Li, M. Nisar, Y. Rong., L. Zhang,: A Process-Agent Construction Method for Software Process Modeling in SoftPM. Software Process Change (2006) 204-213

[6] L.Console, P.Torasso: A spectrum of logical definitions of model-based diagnosis. Computational Intelligence 7 (1991) 133-141

[7] R.Reiter,: A theory of diagnosis from first principles. Vol. 32. Elsevier Science Publishers Ltd. (1987) 57-95

[8] C.Witteveen, N.Roos, R.v.d.Krogt, M.d.Weerdt,: Diagnosis of single and multi-agent plans. International Conference on Autonomous Agents ACM (2005) 805 – 812

[9] Q. Wang, M. Li,X Liu,: An Active Measurement Model for Software Process Control and Improvement. Journal of Software 16 (2005) 407-418

[10] Q. Wang, M. Li,: Software Process Management: Practices in China. International Software Process Workshop. Springer (2005) 317-341

[11] J. Xiao, Leon J. Osterweil, L. Zhang, A., Wise, Q. Wang.: Applying Little-JIL to describe Process-Agent knowledge and support project planning in SoftPM: Research Sections. Vol. 12. John Wiley and Sons Ltd. (2007) 437-448

[12] N.J.Nilsson, R.E. Fikes: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 5 (1971) 189-208

[13] H.Tonino, A.Bos, M.de Weerdt, C.Witteveen: Plan coordination by revision in collective agent based systems. Artificial Intelligence 142 (2002) 121-145

[14] N. Roos, A. ten Teije, C.Witteveen: A protocol for multi-agent diagnosis with spatially distributed knowledge. Proceedings of the second international joint conference on Autonomous agents and multiagent systems. ACM, Melbourne, Australia (2003)

Mobile-FIRST: A Mobile Agent Based FIrst Responder SysTem

Jason Honda
Harry H. Cheng*
Integration Engineering Laboratory
University of California, Davis
{jmhonda, hhcheng}@ucdavis.edu

Donna Djordjevich
Exploratory Computer and Software Engineering
Sandia National Laboratories
dddjord@sandia.gov

Abstract

There is a growing demand for first responder training applications. The ability to dynamically change the application is critical, but there are currently no applications that add this kind of flexibility and have the power and ease of use that is necessary. In this paper, we propose Mobile-FIRST, a Mobile Agent Based First Responder System, that uses mobile agents in order to add much needed flexibility to first responder training applications. Using a mobile agent system for managing migration and execution of mobile agents which can interface with the running application, we can achieve a flexible and powerful system. Different architectures that take advantage of a mobile agent based system are presented in the paper. A case study is then examined where we have integrated our mobile agent system into a first responder training video game currently in development at Sandia National Laboratories.

1. Introduction

In the world of first responder training, flexibility is important. There is a growing demand for training applications and flexibility makes the difference between a valuable or useless application. The ability to change and create new scenarios is critical. Being able to change the training scenario from something general to something not routine is important in training[1]. The ability to easily script out all of these changes and execute them dynamically would be incredibly powerful. Being able to change a running application can also be beneficial. An interpreted mobile agent solution could help to solve these problems.

The current technologies for changing an application are lacking. An area of interest in changing an application is scripting in video games[2]. There are also proprietary scripting languages used by developers that are application specific, such as Linden Scripting Language (LSL) used in Second Life[3]. The problem with these is that they are usually application specific and do not have a great amount of power to change the application. There are also more general programming languages used for scripting such as Python[4] and Lua. These scripting languages also do not have the ability to change a running application.

Another way that we can change an application is to rewrite the application. We can of course go into the core application code and add features, add/change scenarios, and then recompile the application. This takes a lot of effort and also requires expert programmers that have access to the source code and also intimate knowledge of the application itself. This is impractical as a continued model of adding to and tweaking an application.

Another area of research that is related is that of dynamic software updating[5]. The way that this works is that we can swap certain things at runtime such as types, classes, or objects but usually involving simple definitions. This is mainly designed for long running applications that can not afford to have downtime and may need to be updated or fixed on the fly.

There is also edit-and-continue technology where you can change certain aspects of the program and compile it while the program is running and relink to the new code[6]. This is usually tied to an integrated development environment. This is designed to be used in the development process in order to make changes quickly, without recompiling and running. Edit-and-continue is not ideal for a stable release application.

In this paper we propose adding a mobile agent based system to first responder training applications. Using an embedded interpreter that can interface with the running binary application and a mobile agent system for mobile code agents to migrate and execute in these applications, we can achieve a flexible and powerful mobile agent based system.

This paper is broken down as follows. In sections 2 and 3, Mobile-FIRST, a Mobile Agent Based First Responder System, is introduced and the different architectures that we can use with it are explored. Section 4 discusses how we implemented the Mobile-FIRST library and how applications can use it. In section 5, we look at a case study integrating

*Address all correspondance to this author

the mobile agent based system into a first responder training video game in development, GroundTruth. In section 6 we conclude the paper by summarizing the results.

2. Mobile-FIRST

Agents are program entities that execute independently and are impacted by the environment and have control over themselves and the environment. Creating large software systems using agents has been explored before. The nature of many kinds of software including first responder training applications lends nicely to agent-based encapsulation of modules. Agents could represent entities in the application that have changing behaviors and locations, such as first responders, traffic, and population entities.

A mobile agent is an agent that can migrate to different machines and begin executing. It can stop executing, migrate again, and continue executing. Adding mobile agents will provide us with greater power and flexibility in altering an application. These mobile agents can represent any number of things and can come from anywhere, peers playing a game in a multiplayer fashion, a server that stores scenarios and scripts, or a designated "red team" that can change the application while running and altering the scenario to increase or decrease difficulty. Being mobile agents, they can communicate with other agents on a system, as well as migrate in order to accomplish whatever task it is doing.

Mobile-FIRST encompasses a mobile agent system and interpreter along with Mobile-FIRST libraries. Mobile-FIRST provides applications with the ability to have mobile agents being executed within an application interfacing with the binary space and changing the running application.

2.1. Mobile Agent System

The mobile agent system used in Mobile-FIRST is Mobile-C[7] developed at the Integration Engineering Laboratory at University of California, Davis. Mobile-C[8][9] is a mobile agent platform for C/C++ agents. Agents are encapsulated in XML and are sent to other machines. Upon arrival they are executed in an interpreter. Mobile-C executes agents in an Embedded Ch[10] interpreter which can interface with the binary application space. Mobile-C is a full featured mobile agent system and is fully accessible by Mobile-FIRST.

2.2. Benefits

One benefit of using a mobile agent based system is that we can write our scripts in C/C++. This is a more familiar language that most people know and can program in. This means that there is little or no learning curve for most developers, and it also facilitates an easier transition from script code writers to developers since they are not writing in a watered down proprietary scripting language.

By providing the proper interface to binary space for the interpreted agent space, an agent could do anything that could have been done hard coding the program. This provides a great amount of power to affect the application in any way that we can think of through the given API.

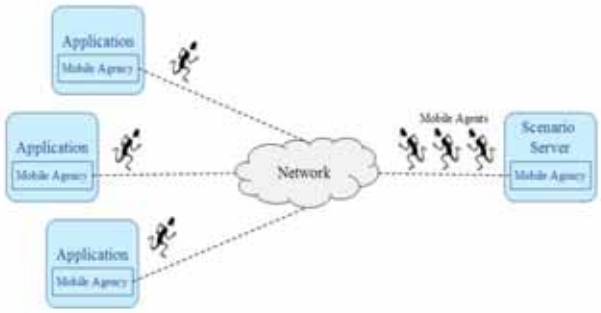


Figure 1. Client-Server Architecture

3. Architectures in Mobile-FIRST

There are many possible architectures that we can use with a mobile agent based system. These different architectures can be used in a wide variety of applications. An application might use a combination of different architectures at the same time in order to accomplish differing objectives with Mobile-FIRST.

The application will use Mobile-FIRST which contains Mobile-C and Embedded Ch interpreters. Mobile-C will wait for incoming agents and when one arrives, it will initialize it in its own interpreter that has access to a library of functions in the binary application space.

3.1. Client-Server Architecture

The first way that we can use a mobile agent system is in a client-server manner as shown in Figure 1. One application of this architecture could be designated servers that store agents for varying scenarios or modules that can be requested at execution time of the application, retrieving these agents and executing them. In an architecture such as this the client would send a request agent that will migrate to the server where it can intelligently request the needed agent to be sent back. An application does not need all possible agents and scripts, it can at runtime intelligently download the needed agents from one central server. This could also be used when security is an issue and a client can run an agent but should not be storing these agents so they get agents from a secure server.

Another application is a more standard mobile agent application where an agent can migrate from a client machine to the server to process something that is either more efficient to run on a server or can only be ran on a server. This could be computationally intensive things that can be computed remotely and the results sent back or maybe proprietary algorithms that are sensitive and need to be protected and only run and reside on a secure server.

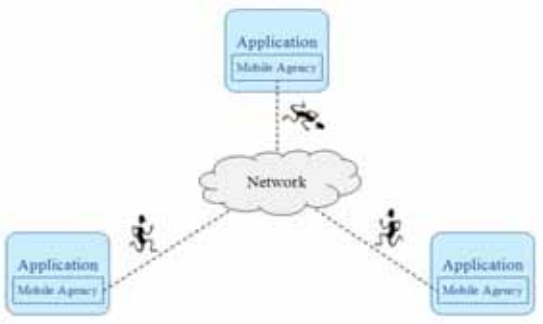


Figure 2. Peer-to-peer Architecture

3.2. Peer-to-peer Architecture

We could also use a peer-to-peer architecture in a multiplayer environment as shown in Figure 2 where there are multiple clients both sending and receiving agents from other connected clients. The actions of one player could affect the others in new and unplanned ways. One client could have an agent migrate to other clients and change their application. There are many ways that this can be used where many people are running an application and agents are migrating from user to user affecting each other, sharing data, or sharing computing resources.

3.3. Red Team Architecture

Another architecture that could be used is a "red team" architecture. A "red team" is an adversary group that is monitoring a trainee and would have the ability to ramp up or decrease the difficulty in order to test them appropriately. This architecture would resemble a peer-to-peer architecture where there are two connected clients with one affecting the other, and the trainee seemingly unaware of the "red team".

With the mobile agent architecture they could at runtime inject agents into the application that could increase difficulty such as setting up another incident that the trainee might have to respond to, or they could send in additional resources to help the player out. There are many things that can be done here and is only limited by what can be written in scripts from changing the difficulty to advanced monitoring of user actions for later assessment or omniscient knowledge of their actions in order to react in an adversary manner quickly.

4. Implementation

Mobile-FIRST is currently being developed as a library that can easily be added into any project. Mobile-FIRST handles all aspects of including mobile agents into the project. The library handles creating and destroying the mobile agent server. As agents arrive at the machine, it provides a mechanism for arriving agents to register themselves as a module for later use by the application or register to be ran at a certain time interval or at a certain time in the future, providing flexibility for the agent to choose how to run. By having the agents register themselves, it becomes known what these agents represent and what functions these agents will handle. Other classes in the application can then access the mobile agents through the Mobile-FIRST class and then call functions on the agents.

4.1. Exposing Classes to Interpreted Space

We want agents to be able to change objects and call functions in the application. In order for the interpreted agent space to be able to interact with the binary application space, the C++ classes in the application need to be exposed to interpreted application space. They need to be written in a way to expose member variables and functions of the class. More information on this process can be found in Embedded Ch documentation.

5. Case Study: GroundTruth

Ground Truth is a first responder training video game developed by Sandia National Laboratories and USC GamePipe. It is a 3D real time strategy video game for training first responders on how to handle emergency situations from the incident commander point of view. The current game is written in C++ and uses the OGRE 3D open source graphics engine.

The game is designed to have fast paced scenarios that stress the user into making quick, correct decisions. The initial scenario is a large scale chemical release in the middle of a densely populated city. Proper use of resources is necessary to mitigate harm to the public. This can be done by having police and fire evacuate or shelter in place buildings, direct traffic, and rescue people.

5.1. Mobile-FIRST Integration

A simple interface into the binary space of the GroundTruth game has been created to test the concept of sending agents with Mobile-FIRST. This simple interface wraps some basic functionality of the game that is immediately visible in a running game. These features include logging, alert pop-ups, and killing of the population. The mobile agent coded in XML in Program 1 was used to produce the effects seen in Figure 3. Upon arrival, the agent dynamically added an incident into the running game, killed some of the population, and notified the player. This is an example of a "red-team" architecture as presented in 3.3 where a "red-team" has sent in an agent that applies an incident dynamically in order to increase the difficulty for the trainee by having to deal with another incident.

As you can see in the Figure 3, there are many apparent things that happened when the agent was sent over. An alert popped up alerting the user of a new situation, along with the population being affected.

Work was then done to replicate a more realistic application involving abstracting out the algorithm that affects population. This involved exposing whole class interfaces to the mobile agent space. By exposing the class that manages the population and giving the population algorithm

```
<?xml version="1.0"?>

<!DOCTYPE myMessage SYSTEM "gafmessage.dtd">

<GAF_MESSAGE>
 <MESSAGE message="MOBILE_AGENT">
  <MOBILE_AGENT>
   <AGENT_DATA>
    <NAME>gtagent</NAME>
    <OWNER>IEL</OWNER>
    <HOME>localhost:5051</HOME>
    <TASK task="1" num="0">
      <DATA
           persistent="1"
           number_of_elements="0"
           name="no-return"
           complete="0"
           server="localhost:5051">
     </DATA>
    <AGENT_CODE>
     <![CDATA[
#include <chscript.h>
int main()
{
  logMessage("Killing 100 people");
  createFullAlert("A gas main just exploded killing 100 people", \
    "Police on scene", "Police", "Red", 35, 35);
  int i;
  for(i = 0; i<100; i++){
    killPerson();
  }
  return 0;
}
     ]]>
    </AGENT_CODE>
   </TASK>
  </AGENT_DATA>
 </MOBILE_AGENT>
</MESSAGE>
</GAF_MESSAGE>
```

Program 1. A mobile agent with mobile code

Figure 3. Effects of a simple agent with enlarged visuals

agent a pointer to this class, the agent can at will affect the population. The agent can choose how often to run, the severity at which to affect the population and the criteria for which people are affected.

Work was done to seamlessly transition from a binary algorithm to the agent algorithm upon arrival. Using the Mobile-FIRST libraries, the application can check to see if an agent is available for a certain module, and then start calling functions on the agent, or if not present, it can send an agent to intelligently request an agent for this module from a server, as in the client-server architecture presented in 3.1

Many benefits can be seen with using the mobile agent system in this application. We gain the ability to swap modules at run time. Using mobile agents allows for intelligent agents to be dynamically added to the running application and change it in new and unforseen ways interacting with the binary space.

6. Conclusion

Mobile-FIRST, a mobile agent based first responder system, has been presented in this paper. Three architectures using this system have been outlined. Mobile-FIRST has been implemented and verified with GroundTruth, a Sandia National Laboratories first responder training application. Using Mobile-FIRST and mobile agents provides a powerful solution to adding dynamic behavior to a first responder application. Using an embedded C/C++ interpreter to execute the mobile code, these agents provide a great amount of power to interface to the binary space of the running application at runtime. Modules and features can be added at runtime and can change the running appliction. Integrating Mobile-FIRST into an application provides us with the ability to move agents from foreign servers or other peers using the application in a cooperative manner or from others playing in an adversarily manner.

Mobile-FIRST could provide a whole new way of looking at how to create dynamic behavior in an application as well as how certain applications are architected. It also provides the ability to integrate easily into real world deployments that interface with many other devices.

References

[1] J. K. Ford and A. M. Schmidt, "Emergency response training: strategies for enhancing real-world performance," *Journal of Hazardous Materials*, vol. 75, no. 2-3, pp. 195–215, Jun 2000.

[2] A. M. Phelps and D. M. Parks, "Fun and games: Multi-language development," *Queue*, vol. 1, no. 10, pp. 46–56, 2004.

[3] *LSL Portal - Second Life Wiki*, Linden Lab, wiki.secondlife.com/wiki/LSL_Portal.

[4] B. Dawson, "Game scripting in python," in *Game Developers Conference Proceedings*, 2002.

[5] M. Hicks and S. Nettles, "Dynamic software updating," *ACM Trans. Program. Lang. Syst.*, vol. 27, no. 6, pp. 1049–1096, 2005.

[6] M. Eaddy and S. Feiner, "Multi-language edit-and-continue for the masses," *Tech Rep CUCS-015-05. Dept. of CS, Columbia Univ.*, Apr 2005.

[7] *Mobile-C: A Multi-agent Platform for Mobile C/C++ Code*, http://www.mobilec.org.

[8] B. Chen, H. H. Cheng, and J. Palen, "Mobile-c: a mobile agent platform for mobile c/c++ agents," *Software: Practice and Experience*, vol. 36, no. 15, pp. 1711–1733, 2006.

[9] B. Chen, D. D. Linz, and H. H. Cheng, "Xml-based agent communication, migration and computation in mobile agent systems," *Journal of Systems and Software*, 2007, doi:10.1016/j.jss.2007.10.026.

[10] *Embedded Ch*, SoftIntegration, Inc., http://www.softintegration.com/products/sdk/embedch/.

Ontology-based and Evolutionary Search for Computational Agents Schemes

Roman Neruda*

Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodarenskou vezi 2, 18207 Prague 8, Czech Republic
roman@cs.cas.cz

Abstract

This work deals with a problem of automatic composition of multi-agent system satisfying given constraints. A general concept of representation of connected groups of agents (schemes) within a multi-agent system is introduced and utilized for automatic building of schemes to solve a given computational intelligence task. We propose a combination of an evolutionary algorithm and a formal logic resolution system which is able to propose and verify new schemes. The approach is illustrated on simple examples.

1 Introduction

Autonomous agents are small self-contained programs that can solve simple problems in a well-defined domain [10]. In order to solve complex problems, agents have to collaborate, forming Multi-Agent Systems (MAS). A key issue in MAS research is how to generate MAS configurations that solve a given problem [5]. In most Systems, an intelligent (human) user is required to set up the system configuration. Developing algorithms for automatic configuration of Multi-Agent Systems is a major challenge for AI research.

We have developed a platform for creating Multi-Agent Systems [7], [9]. Its main areas of application are computational intelligence methods (genetic algorithms, neural networks, fuzzy controllers) on single machines and clusters of workstations. Hybrid models, including combinations of artificial intelligence methods such as neural networks, genetic algorithms and fuzzy logic controllers, seem to be a promising and extensively studied research area [2]. Our distributed multi-agent system — provides a support for an easy creation and execution of such hybrid AI models utilizing the Java/JADE environment.

The above mentioned applications require a number of cooperating agents to fulfill a given task. So far, MAS are created and configured manually. In this paper, we introduce two approaches for creation and possible configuration of MAS. One of them is based on formal descriptions and provides a logical reasoning component for the system.

The second approach to MAS generation employs evolutionary algorithm (EA) which is tailored to work on special structures—directed acyclic graphs—denoting MAS schemata. The advantage of EA is that it requires very little additional information apart from a measure of MAS performance. Thus, the typical run of EA consists of thousands of simulations which build and assess the fitness values of various MAS. Since the properties of logical reasoning search and evolutionary search are dual, the ultimate goal of this work is to provide a solution combining these two approaches in a hybrid search algorithm. This paper presents the first steps towards such a goal.

2 Description of MAS by means of Logics

The most natural approach to formalize ontologies is the use of First Order Predicate Logics (FOL). The disadvantage of FOL-based languages is the expressive power of FOL. FOL is undecidable [4], and there are no efficient reasoning procedures. Nowadays, the de facto standard for ontology description language for formal reasoning is the family of description logics. Description logics are equivalent to subsets of first order logic restricted to predicates of arity one and two [3]. They are known to be equivalent to modal logics [1]. For the purpose of describing multi-agent systems, description logics are sometimes too weak. In these cases, we want to have a more expressive formalism. We decided to use Prolog-style logic programs for this. In the following we describe how both approaches can be combined together.

An *agent* is an entity that has some form of perception of its environment, can act, and can communicate with other agents. It has specific skills and tries to achieve goals. A *Multi-Agent System (MAS)* is an assemble of interacting agents in a common environment [6]. In order to use automatic reasoning on a MAS, the MAS must be described in formal logics. For the computational system, we define a formal description for the static characteristics of the agents, and their communication

*This research has been supported by the the project 1ET100300419 of the Program Information Society (of the Thematic Program II of the National Research Program of the Czech Republic) "Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realization".

channels. We do not model dynamic aspects of the system yet.

Agents communicate via messages and triggers. Messages are XML-encoded FIPA standard messages. Triggers are patterns with an associated behavior. When an agent receives a message matching the pattern of one of its triggers, the associated behavior is executed. In order to identify the receiver of a message, the sending agent needs the message itself and an id of the receiving agent. A conversation between two agents usually consists of a number of messages conforming to FIPA protocols. In order to abstract from the actual messages, we subsume all these messages under a *message type* when describing an agent in formal logics.

Definition 1 (Message type) *A* message type *identifies a category of messages that can be send to an agent in order to fulfill a specific task. We refer to message types by unique identifiers.*

The set of message types understood by an agent is called its *interface*. For outgoing messages, each link of an agent is associated with a message type. Via this link, only messages of the given type are sent. We call a link with its associated message type a *gate*.

Definition 2 (Interface) *An interface is the set of message types understood by a class of agents.*

Definition 3 (Gate) *A gate is a tuple consisting of a message type and a named link.*

Now it is easy to define if two agents can be connected: Agent A can be connected to agent B via gate G if the message type of G is in the list of interfaces of agent B. Note that one output gate sends messages of one type only, whereas one agent can receive different types of messages. This is a very natural concept: When an agent sends a message to some other agent via a gate, it assigns a specific role to the other agent, e.g. being a supplier of training data. On the receiving side, the receiving agent usually should understand a number of different types of messages, because it may have different roles for different agents.

Definition 4 (Connection) *A connection is described by a triple consisting of a sending agent, the sending agent's gate, and a receiving agent.*

Next we define *agents* and *agent classes*. Agents are created by generating instances of classes. An agent derives all its characteristics from its class definition. Additionally, an agent has a name to identify it. The static aspects of an agent class are described by the interface of the agent class (the messages understood by the agents of this class), the gates of the agent (the messages send by agents of this class), and the type(s) of the agent class. Types are nominal identifiers for characteristics of an agent. The types used to describe the characteristics of the agents should be ontological sound.

Concepts	
mas(C)	C is a Multi-Agent System
class(C)	C is the name of an agent class
gate(C)	C is a gate
m_type(C)	C is a message type
Roles	
type(X,Y)	Class X is of type Y
has_gate(X,Y)	Class X has gate Y
gate_type(X,Y)	Gate X accepts messages of type Y
interface(X,Y)	Class X understands mess. of type Y
instance(X,Y)	Agent X is an instance of class Y
has_agent(X,Y)	Agent Y is part of MAS X

Table 1. Concepts and roles used to describe MAS.

```
class(decision_tree)
type(decision_tree, computational_agent)
has_gate(decision_tree, data_in)
gate_type(data_in, training_data)
interface(decision_tree, control_messages)
```

Figure 1. Example agent class definition.

Definition 5 (Agent Class) *An agent class is defined by an interface, a set of message types, a set of gates, and a set of types.*

Definition 6 (Agent) *An agent is an instance of an agent class. It is defined by its name and its class.*

A Multi-Agent System can be described by three elements: The set of agents in the MAS, the connections between these agents, and the characteristics of the MAS. The characteristics (constraints) of the MAS are the starting point of logical reasoning: In *MAS checking* the logical reasoner deduces if the MAS fulfills the constraints. In *MAS generation*, it creates a MAS that fulfills the constraints, starting with an empty MAS, or a manually constructed partial MAS.

Definition 7 (Multi-Agent System) Multi-Agent Systems (MAS) *consist of a set of agents, a set of connections between the agents, and the characteristics of the MAS.*

In order to describe agents and Multi-Agent Systems in description logics, the definitions 1 to 7 are mapped onto description logic concepts and roles as shown in table 1. An example agent class description is given in figure 1. It defines the agent class "decision_tree". This agent class accepts messages of type "control_message". It has one gate called "data_in" for data agent and emits messages of type "training_data".

In the same way, A-Box instances of agent classes are defined: $instance(decision\_tree, dt\_instance)$ An agent is assigned to a MAS via role "has_agent". In the following example, we define "dt_instance" as belonging to MAS "my_mas": $has\_agent(my\_mas, dt\_instance)$

Since connections are relations between three elements, a sending agent, a sending agent's gate, and a receiving agent,

we can not formulate this relationship in traditional description logics. It would be possible to circumvent the problem by splitting the triple into two relationships, but this would be counter-intuitive to our goal of defining MAS in an ontological sound way. Connections between agents are relationships of arity three: Two agents are combined via a gate. Therefore, we do not use description logics, but traditional logic programs in Prolog notation to define connections: $connection(dt\_instance, other\_agent, gate)$

Constraints on MAS can be described in Description Logics, in Prolog clauses, or in a combination of both. As an example, the following concept description requires the MAS "dt_MAS" to contain a decision tree agent: $dt\_MAS \sqsupseteq mas \sqcap has\_agent.(\exists instance.decision\_tree)$

An essential requirement for a MAS is that agents are connected in a sane way: An agent should only connect to agents that understand its messages. According to definition 4, a connection is possible if the message type of the sending agent's output gate matches a message type of the receiving agent s interface. With the logical concepts and descriptions given in this section, this constraint can be formulated as a Prolog style horn rule. If we are only interested in checking if a connection satisfies this property, the rule is very simple:

connection(S,R,G) ←
    instance(R, RC) ∧
    instance(S, SC) ∧
    interface(RC, MT)∧
    has_gate(SC, G) ∧
    gate_type(G, MT)

The following paragraphs show an example for logical descriptions of MAS. *Computational MAS:* A computational MAS can be defined as a MAS with a task manager, a computational agent and a data source agent which are interconnected.

comp_MAS(MAS) ←
    type(CAC, computational_agent)∧
    instance(CA, CAC)∧
    has_agent(MAS, CA)∧
    type(DSC, data_source)∧
    instance(DS, DSC)∧
    has_agent(MAS, DS)∧
    connection(CA, DS, G)∧
    type(TMC, task_manager)∧
    instance(TMC, TM)∧
    has_agent(MAS, TM)∧
    connection(TM, CA, GC)∧
    connection(TM, DS, GD)

3 Evolutionary search

The proposed evolutionary algorithm operates on schemes definitions in order to find a suitable scheme solving a specified problem. The genetic algorithm has three inputs: First, the number and the types of inputs and outputs of the scheme. Second, the *training set*, which is a set of prototypical inputs and the corresponding desired outputs, it is used to compute the fitness of a particular solution. And third, the list of types of building blocks available for being used in the scheme.

We supply three operators that would operate on graphs representing schemes: *random scheme creation*, *mutation* and *crossover*. The aim of the first one is to create a random scheme. This operator is used when creating the first (random) generation. The diversity of the schemes that are generated is the most important feature the generated schemes should have. The goal of the crossover operator is to create offsprings from two parents. The crossover operator proposed for scheme generation creates one offspring. The operator horizontally divides the mother and the father, takes the first part from father's scheme, and the second from mother's one. The mutation operator is very simple. It finds two links in the scheme (of the same type) and switches their destinations.

4 Experiments

This section describes the experiments we have performed with generating the schemes using the genetic algorithm described above.

The training sets used for experiments represented various polynomials. The genetic algorithm was generating the schemes containing the following agents representing arithmetical operations: *Plus* (performs the addition on floats), *Mul* (performs the multiplication on floats), *Copy* (copies the only input (float) to two float outputs), *Round* (rounds the incoming float to the integer) and finally *Floatize* (converts the int input to the float).

The selected set of operators has the following features: it allows to build any polynomial with integer coefficients. The presence of the *Round* allows also another functions to be assembled. These functions are the 'polynomials with steps' that are caused by using the *Round* during the computation.

The results of the experiments depended on the complexity of the desired functions. The functions, that the genetic algorithm learned well and quite quickly were functions like $x^3 - x$ or x^2y^2. The learning of these functions took from tens to hundred generations, and the result scheme precisely computed the desired function.

Also more complicated functions were successfully evolved. Having in mind, that the only constant that can be used in the scheme is -1, we can see, that the scheme is quite big (comparing to the previous example where there was only approximately 5–10 building blocks) — see Fig. 2. It took much more time/generations to achieve the maximal fitness, namely 3000 in this case.

On the other hand, learning of some functions remained in the local maxima, which was for example the case of the function $x^2 + y^2 + x$.

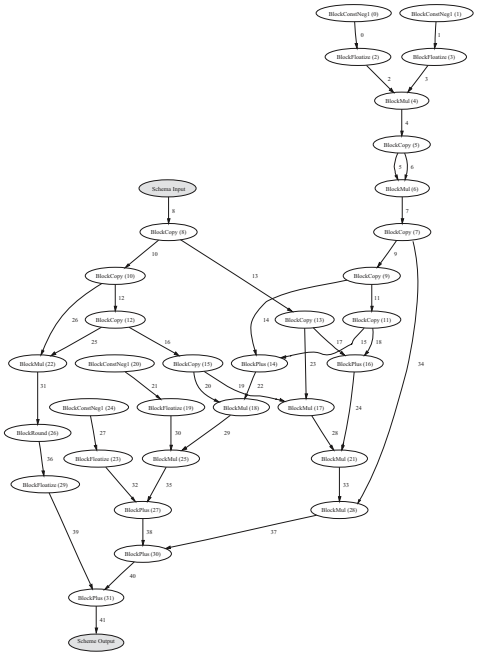


Figure 2. Function $x^3 - 2x^2 - 3$. The scheme with fitness 1000 (out of 1000), taken from 3000th generation.

5 Conclusions

We have presented a hybrid system that uses a combination of evolutionary algorithm and a resolution system to automatically create and evaluate multi-agent schemes. So far, the implementation has focused on relatively simple agents computing parts of arithmetical expressions. Nevertheless, the sketched experiments demonstrate the soundness of the approach. A similar problem is described and tackled in [11] by means of matchmaking in middle-agents where authors make use of ontological descriptions but utilize other search methods than EA.

In our future work we plan to extend the system in order to incorporate more complex agents into the schemes. Our ultimate goal is to be able to propose and test schemes containing a wide range of computational methods from neural networks to fuzzy controllers, to evolutionary algorithms. While the core of the proposed algorithm will remain the same, we envisage some modifications in the genetic operators based on our current experience.

Namely, a finer consideration of parameter values, or configurations, of basic agents during the evolutionary process needs to be addressed. So far, the evolutionary algorithm rather builds the -3 constant by combining three agents representing the constant 1, than modifying the constant agent to represent the -3 directly. We hope to improve this behavior by introducing another kind of genetic operator. This mutation-like operator can be more complicated in the case of real computational agents such as neural networks, though. Nevertheless, this approach can reduce the evolutionary algorithm search space substantially.

We also plan to extend the capabilities of the resolution system towards more complex relationship types than the ones described in this paper. In our work [8] we use ontologies for the description of agent capabilities, and have the CSP-solver reason about these ontologies. The next goal is to provide hybrid solution encompassing the evolutionary algorithm enhanced by ontological reasoning.

References

[1] F. Baader. Logic-based knowledge representation. In M. J. Wooldrige and M. Veloso, editors, *Artificial Intelligence Today, Recent Trends and Developments*, pages 13–41. Springer, 1999.

[2] P. Bonissone. Soft computing: the convergence of emerging reasoning technologies. *Soft Computing*, 1:6–18, 1997.

[3] Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367, 1996.

[4] M. Davis, editor. *The Undecidable—Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Raven Press, 1965.

[5] J. E. Doran, S. Franklin, N. R. Jennings, and T. J. Norman. On cooperation in multi-agent systems. *The Knowledge Engineering Review*, 12(3):309–314, 1997.

[6] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Harlow: Addison Wesley Longman, 1999.

[7] Pavel Krušina, Roman Neruda, and Zuzana Petrova. More autonomous hybrid models in bang. In *International Conference on Computational Science (2)*, pages 935–942, 2001.

[8] R. Neruda and G. Beuster. Towards dynamic generation of computational agents by means of logical descriptions. In *MASUPC'07 – International Workshop on Multi-Agent Systems Challenges for Ubiquitous and Pervasive Computing*, pages 17–28, 2007.

[9] Roman Neruda, Pavel Krušina, Petra Kudova, and Gerd Beuster. Bang 3: A computational multi-agent system. In *Proceedings of the 2004 WI-IAT'04 Conference*. IEEE Computer Society Press, 2004.

[10] H. S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(2):205–244, 1995.

[11] Zili Zhang and Chengqi Zhang. *Agent-Based Hybrid Intelligent Systems*. Springer Verlag, 2004.

A Goal-Oriented Mixed-Granularity Component Selection Method for Huge Component Repositories

Xiaolin Xi Jiyong Park Jiakun Liu Seongsoo Hong
Seoul National University, Korea
sinia@redwood.snu.ac.kr parkjy@redwood.snu.ac.kr jkliu@dsp.snu.ac.kr sshong@redwood.snu.ac.kr

***Abstract*:** *In the component-based software development, component selection is a critical step since the selected components considerably affect the system quality. Traditionally, the selection has been done in an ad-hoc manner, which takes a long time and does not guarantee the quality of the resultant system. These problems become more significant as the complexity of a system or the size of component repositories increase. Although a number of selection methods have been proposed, they cannot ensure the suitability of selected components on system-level and fail to consider mixed-granularity components. We present a goal-oriented mixed-granularity (GOMG) component selection method to solve these problems. It adopts a hierarchical goal tree to provide options to use components in different granularities. Also, it exploits a hierarchical evaluation method that systematically measures the appropriateness of component sets on the system level. We have conducted a case study of building a composite SoC CAD tool. The component repository consisted of 250 components that varied considerably in both granularity and performance. The results show that our GOMG method yields a better aggregated score than the existing method by 33% and reduces the time consumed on the selection by 73%.*

1. Introduction

Component-based software development (CBSD) is an engineering practice used to build a software system by composing software components, which are software artifacts that are specially designed to be used in diverse contexts. A typical CBSD process includes five main steps as shown in Fig.1: requirement analysis, component selection, adaptation, integration and evolution [1]. Among these steps, the selection step is especially important because finding an appropriate component set is a prerequisite for developing a high-quality system.

Traditionally, the selection has been performed in an ad-hoc manner, which takes a long time and the suitability of selected components cannot be ensured as the complexity of systems and the number of available components increase.

In the literature, methods have been proposed to address the component selection problem, e.g., OTSO [2], CEP [3] and CARE [4]. In practice, however, they face many problems. First, since they evaluate each component individually, it is difficult for them to guarantee the optimality of a selected component set on the system level. Second, since they ignore the possibility of selecting components in different granularities, they may fail to find utilizable components.

To tackle these problems, we propose a goal-oriented mixed granularity (GOMG) approach employing two main techniques. First, a goal-oriented requirement analysis method is adopted to decompose the system to be designed into a hierarchy of mixed-granularity sub-systems. Then, a hierarchical evaluation method explores mixed-granularity components for the system and evaluates these combinations from the lowest level to the system level to find out the best component set with respect to the design goals.

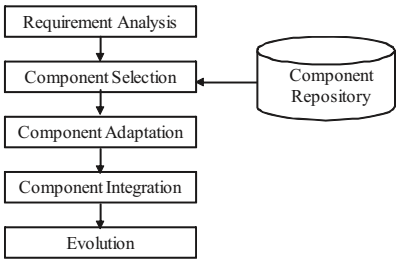


Fig. 1. A typical CBSD process.

The paper is organized as follows. In Section 2, we define the problem and give a solution overview. In Section 3, we present a goal-oriented requirement analysis that derives a hierarchical goal tree. Then, a three-step selection method is explained in Section 4. Section 5 shows a case study to build a composite SoC CAD tool using our approach. Finally, we conclude our work in Section 6.

2. Problem Definition and Solution Overview

In this paper, we are going to solve the problem of how to objectively find a combination of components, possibly with different granularities, that best meets customer requirements. Therefore, our GOMG component selection approach respects the following two important criteria.

- A systematical and objective evaluation method is needed to ensure the optimality of the selected component set on the system level.
- The components in different granularities should be explored.

Our GOMG method meets these criteria through a systematic process as shown in Fig. 2.

- First, the goal-oriented requirement analysis method formalizes customers' unstructured requirements into a goal tree. The goals on each level match components with different granularities. Components in small-granularity, which accomplish low-level goals, can be joined together to form large-granularity components to achieve high-level goals. Hence, options for selecting components in all granularities are provided.

- Second, for each goal in the tree, we obtain candidate components from the repository through search.
- Third, we score the candidate components to measure how well they meet the non-functional requirements.
- Finally, through a hierarchical evaluation, a component set that has highest aggregated score is obtained. The evaluation is done in a bottom-up way, from the lowest-level leaf nodes to the system-level root node. At each node, the component with the highest score is identified from the candidate components. Then the identified components are promoted to the candidate component of their parent goals. This process repeats up to the root system-level goal. As a result, the component set with the best score is selected.

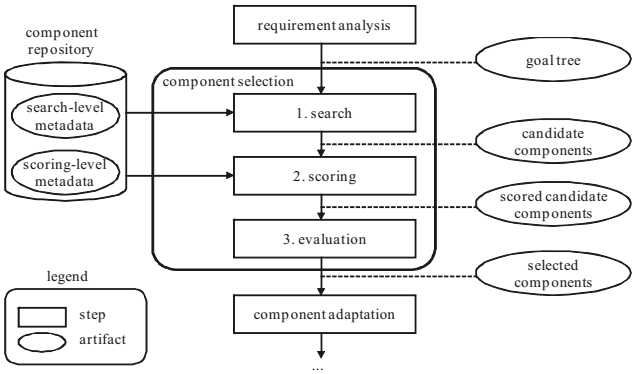


Fig. 2. The overview of GOMG component selection method.

3. Goal-Oriented Requirement Analysis

The requirement analysis transforms a set of informally expressed requirements into detailed, structured specifications. Our approach refers to goal-oriented requirement engineering [5], where goals are defined as objectives the system under consideration should achieve [6]. Goals are categorized into functional goals and non-functional goals. *Functional goals* describe system services or functions, e.g., modulating a signal. *Non-functional goals* are constraints on the system or on the development process, e.g., performance requirements, security or reliability [7]. To illustrate our method, we present a chatting client as a walk-through example.

3.1. The Structure of a Goal Tree

Our goal-oriented requirement analysis transforms the requirements into a hierarchical structure which we call a goal tree. A goal tree is a tree in which a node represents a functional goal and a link represents a dependency between functional goals.

In a goal tree, the top-level functional goal specifies the overall functional objective of the system, while low-level functional goals specify concrete functional objectives of components. An AND-link specifies conjunctive sub-goals which need to be achieved together to fulfill their parent-goal whereas an OR-link specifies disjunctive sub-goals any of which achieves the parent-goal.

A node, hence a functional goal, has non-functional goals as its attributes. The maximum degrees to which the components fulfilling the functional goal can contribute to the non-functional goals are called *contribution factors*. Mathematically, sets of all functional goals and non-functional goals in the system are denoted as $FG = \{fg_1, fg_2 ...\}$ and $NFG = \{nfg_1, nfg_2 ...\}$, respectively. Then the contribution factor of functional goal $fg \in FG$ to non-functional goal $nfg \in NFG$ is derived by function *cf*,

$$cf : FG \times NFG \rightarrow \{x \mid 0 \le x \le 1, x \in \Re\}. \quad (1)$$

3.2. Deriving a Goal Tree

Our method derives a goal tree by breaking down the top-level functional goal into more concrete sub-goals in child nodes. It keeps breaking down the goals until no more levels of details are needed.

The contribution factor in the root node is defined to be '1'.

$$\forall nfg : cf\left(fg, nfg\right) = 1 \quad \text{if } fg \text{ is a top-level goal.} \quad (2)$$

The contribution factors in the internal and leaf nodes are derived by splitting and inheriting the contribution factor from their parent node as in equation (3). AND-linked child nodes split the contribution factor of their parent node. OR-linked child nodes directly inherit the contribution factor from their parent node.

$$\forall nfg : \sum_{fg_c \text{ is a child of } fg_p} cf\left(fg_c, nfg\right) = cf\left(fg_p, nfg\right)$$
$$\text{if } fg_p \text{ is a parent of an AND-linked child node } fg_c, \quad (3)$$
$$\forall nfg : cf\left(fg_c, nfg\right) = cf\left(fg_p, nfg\right)$$
$$\text{if } fg_p \text{ is a parent of an OR-linked child node } fg_c.$$

The goal tree of our walk through example is shown in Fig. 3 and the descriptions of the goals are listed in Table 1. Contribution factors in the root node *fg1* are all '1'. These contribution factors are then split into child goals since they are AND-linked. For example, *cf*(*fg1*, *nfg2*) is split into 1, 0 and 0 to *cf*(*fg1.1*, *nfg2*), *cf*(*fg1.2*, *nfg2*) and *cf*(*fg1.3*, *nfg2*), respectively. On the other hand, in node *fg1.3*, contribution factors are inherited by its OR-linked child nodes.

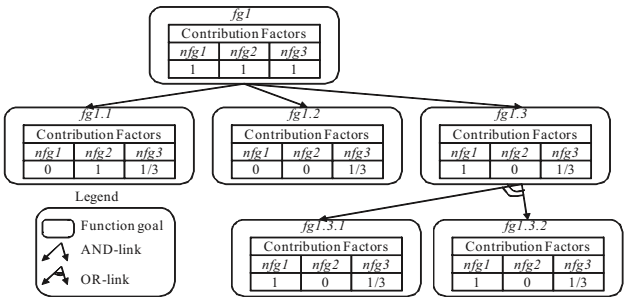


Fig. 3. An example goal tree for a simple chatting client.

The portion of a contribution factor each child node gets depends on the child functional goal's contribution to the non-functional goal. For example, 'minimize the cost' (*nfg3*) is relevant to all components; therefore, the contribution

factor is evenly split by all the child nodes. However, for *nfg2*, the contribution factor of *fg1* is entirely given to *fg1.1* as only the goal 'send message' contributes to satisfying 'average send response time ≤10ms'.

Table 1. Goal description list.

Goal	Description
fg1	Develop a simple chatting client-side system.
fg1.1	Send messages.
fg1.2	Receive messages.
fg1.3	Use networks protocols.
fg1.3.1	Use UDP protocol.
fg1.3.2	Use TCP protocol.
nfg1	Networks reliability, ensure data integrity.
nfg2	Minimize send response time, average time ≤ 10ms.
nfg3	Minimize the cost, cost ≤ 500 USD.

4. GOMG Component Selection Method

The GOMG component selection method is a three-step process: search, scoring and evaluation. In the search step, for each node in the goal tree, it retrieves a group of candidate components. In the scoring step, it scores all the candidate components using the customer provided scoring functions. In the evaluation step, for each goal, it selects a component with the largest aggregated score from the candidate group of a node and promotes it to a new candidate of the parent node. This process repeats from the bottom lowest level up to the top level. As a result, the root node eventually contains the best candidate for the system-level goal.

4.1. Step 1: Search

In the search step, candidate components for each functional goal are retrieved from the component repository through keyword search. Specifically, nouns and verbs are extracted from descriptions of a functional goal and used as keywords. In a component repository, a component is stored with search level metadata that consists of its interface names, operation names and descriptions. The set of candidate components retrieved for a functional goal forms a candidate group of the goal.

4.2. Step 2: Scoring

After finding candidate groups for all nodes in the goal tree, we give scores to each candidate component so that the suitability of a component for goals is clearly quantified. The scoring functions for functional and non-functional goals are defined respectively as below:

$$\begin{aligned} score_f &: CC \times FG \to \{0,1\} \\ score_{nf} &: CC \times NFG \to \{x \mid 0 \le x \le 1, x \in \Re\} \end{aligned} \tag{4}$$

where CC is a set of all the candidate components.

For functional goals, we use function $score_f$ in (4) where $score_f(cc,fg)=0$ implies that candidate component *cc* does not satisfy goal *fg* and $score_f(cc,fg)=1$ denotes that *cc* satisfies *fg*.

For non-functional goals, given a pair of (*cc*, *nfg*), function $score_{nf}$ translates *cc*'s measurement for *nfg* to a real number within interval [0, 1]. The measurements, such as reliability, speed, size, memory footprint and cost, are recorded as the scoring level metadata in the component repository.

4.3. Step 3: Evaluation

After scoring, we use a hierarchical evaluation method to select a set of components that has the highest aggregated score. It is performed in three steps. First, we quantify the relative importance of the non-functional goals by using the concept of an importance factor. Second, we use a weighted sum to derive the aggregated scores for all candidate components. Third, for each goal, we select a component with the largest aggregated score from the candidates of a node and promote it to a new candidate of the parent node. This process repeats from the leaf nodes up to the root node. In the following subsections, we explain these steps in detail.

4.3.1. Reflecting the Tradeoffs among Non-Functional Goals

Non-functional goals have tradeoffs among them. Therefore, we use a concept of *importance factors* to quantify their relative importance. Importance factor if_{nfg} of non-functional goal *nfg* is defined as its share in influencing the customer's satisfaction. Table 2 shows the importance factors for the simple chatting client example.

Table 2. Importance factors for the simple chatting client.

Non-functional goal	*nfg1*	*nfg2*	*nfg3*
Importance factor	0.2	0.2	0.6

Importance factors are used to adjust the contribution factors so that they also reflect the relative importance of non-functional goals. These adjusted contribution factors are called weights. At each node, each contribution factor is multiplied by importance factor if_{nfg} and denoted as

$$weight(fg, nfg) = cf(fg, nfg) \cdot if_{nfg}. \tag{5}$$

This weight reflects node *fg*'s contribution to the customer's satisfaction through non-functional goal *nfg.*

4.3.2. Calculating Aggregated Scores

Since a component has multiple scores for non-functional goals, it is difficult to compare the component with other components. Therefore we derive a single score, called an aggregated score, for each component. Aggregated score for candidate component *cc*, which is in functional goal *fg*'s candidate group, is calculated by (6).

$$score_{agg}(cc, fg) = \begin{cases} 0 & \text{,if } score_f(cc, fg) = 0, \\ \sum_{nfg} weight(fg, nfg) \cdot score_{nf}(cc, nfg), & \text{otherwise.} \end{cases} \tag{6}$$

If component *cc* cannot satisfy a functional goal, its aggregated score is defined as 0. This is because satisfying

the functional goals is the most fundamental requirement. Otherwise, the aggregated score is defined as the weighted sum of all the non-functional goal scores.

4.3.3. Traversing the Goal Tree from Bottom to Top

After the aggregated score for each candidate component is obtained, we compare the aggregated scores while traversing the goal tree from bottom to top. In this process, for each node, the candidate component with the highest aggregated score is identified. The identified component then becomes a candidate for the parent node via promotion. The promotion works differently depending on whether the node is an AND-linked child or an OR-linked child.

For AND-linked child nodes, the candidate components with the best aggregated score are grouped into a composite component and promoted to their parent goal's candidate component. When sub-components are independent, the aggregated score of the composite component is simply defined as the sum of the aggregated scores of the sub-components as follows,

$$score_{agg}(cc, fg) = \sum_{c \text{ is a sub-component of } cc} score_{agg}(c, fg). \quad (7)$$

For the case in Fig. 4, *cc2*, *cc3*, *cc4* are the candidate components with the highest aggregated score for AND-linked child goals *fg1.1*, *fg1.2* and *fg1.3*, respectively. Therefore, the composite candidate component *cc(2+3+4)* is promoted to a candidate component for *fg1*.

For OR-linked child nodes, the candidate component with the best aggregated score is directly promoted to its parent goal's candidate component. For instance, in Fig. 4, *cc4* and *cc5* are promoted to candidate components for *fg1.3* since they have the highest aggregated scores in the OR-linked child goals, *fg1.3.1* and *fg1.3.2*, respectively.

Finally, on the top-level functional goal, the component set with the highest aggregated score is selected. Fig. 4 shows the process of the evaluation.

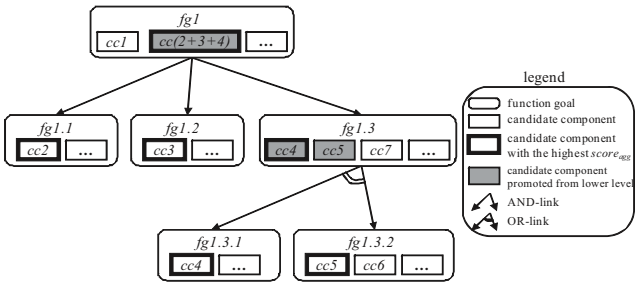


Fig. 4 Evaluation of candidate components for the chatting client

5. Experimental Evaluation

In order to validate the proposed GOMG component selection method, we conducted a case study of building a composite CAD tool for System-on-Chip (SoC) design. We built a component repository with 250 components in this field that varied significantly in granularity, performance and cost. A domain expert was invited to select components using both our method and the existing method, OTSO. We compared the two methods by measuring the aggregated scores and the time consumed for component selection.

Requirements for the SoC tool to develop were given by the customers. With those requirements, we then derived a goal tree by our goal-oriented requirement analysis. After the requirement analysis, the domain expert performed component selection using the two methods. Under OTSO, we tried to select one component at a time that fulfills one of the goals of the system. The domain expert picked a component for each goal and repeated it in a trial-and-error manner until all the selected components satisfied the system-level goal. In contrast, in our method, the domain expert systematically selected a set of components. Our method allowed him to select multiple components simultaneously using the hierarchical goal tree.

As shown in Table 3, the aggregated score of our method yields a better aggregated score than OTSO by 33% and also reduces the time consumed by 73%.

Table 3. Comparison of the GOMG method and OTSO.

Metric	GOMG method	OTSO
Aggregated score	0.812	0.61
Time consumed on selection (man-hour)	15	56

6. Conclusion

We have presented the GOMG component selection method which allows developers to select the best possible components in mixed granularities from enormous component repositories. Our method was evaluated through a case study. The results clearly show that it outperforms OTSO in terms of the overall quality and the time consumed for component selection. As for future study, we will improve the evaluation by considering the influence of sub-components interoperations on composite components' non-functional scores. We are also attempting to perform more extensive case studies to assess its effectiveness. The results look promising.

References

[1] A.W. Brown, K.C. Wallnan, "Engineering of component-based systems," *iceccs*, p. 414, Second IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'96), 1996.

[2] Kontio, J. "A case study in applying a systematic method for COTS selection," *18th International Conference on Software Engineering Berlin*, 1996.

[3] Philips, B., Polen, S, "Add Decision Analysis to Your Cots Selection Process", *Software Technology Support Center Crosstalk*, April 2002.

[4] Chung, L., and Cooper, K., "Knowledge based COTS aware requirements engineering approach," Proc. 14th Int. Conf. Software Eng. Knowledge Eng., 2002, (ACM Press), pp. 175–182.

[5] Alves, C., "Challenges in CoTS-making: A goal-driven requirements engineering perspective", Workshop on Software Engineering Decision Support, Ischia, Italy, July 2002.

[6] Van Lamsweerde, "Goal-oriented requirements engineering: A guided tour", *Proceedings of the IEEE International Conference on Requirements Engineering*, 2001, pp. 249-261.

[7] Ian Sommerville, *Software Engineering*, 6th edition, Addison-Wesley, 2000.

A Case Study: Self-Managed COTS Component-Based Elevator System

Michael E. Shin
Department of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
(806) 742-3527
Michael.Shin@ttu.edu

Fernando Paniagua
Department of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
(806) 742-3527
Fernando.Paniagua@ttu.edu

Abstract

This paper describes a case study for the elevator system, which is implemented using self-managed COTS components. Each COTS component for the elevator system is encapsulated in a wrapper that provides the properties of self-management. By consideration of reusability, application-independent algorithms for the wrapper for COTS components of the elevator system is implemented separately from the application-dependent information so that the application-independent algorithms can be reused for other COTS components in different applications.

1. Introduction

Although COTS (commercial off-the-shelf) components are getting used more and more for many applications, COTS components may still have design faults or unexpected events resulting in system failures. Self-management [Thomas00, Anderson03, Koopman03, IBM03, Dashofy02, Garlan02, Garlan03, Guerra04, Meulen05, Shin06, Shin07] of COTS components has been considered to improve the reliability of COTS component-based systems. Self-managed COTS components detect anomalies in the COTS components, reconfigure the system against the anomalies detected, and repair the anomalies at runtime using a self-managing mechanism encapsulated in a wrapper.

This paper describes a case study for developing a self-managed COTS component-based elevator system. By consideration of reusability, the objects of a wrapper for COTS components are implemented into application-independent algorithms and application-dependent information.

This paper begins by describing self-managed COTS components in section 2. Section 3 describes our approach to implementing self-managed COTS components. Section 4 describes the self-managed elevator system. Section 5 concludes this paper.

2. Self-Managed COTS components

Using a wrapper, a self-managed COTS component implements the properties of self-management. The wrapper is structured into COTS Monitor, COTS Modified Interface, Wrapper Controller, Reconfiguration Manager, and Repair Manager objects (Fig. 1) [Shin6, Shin7].

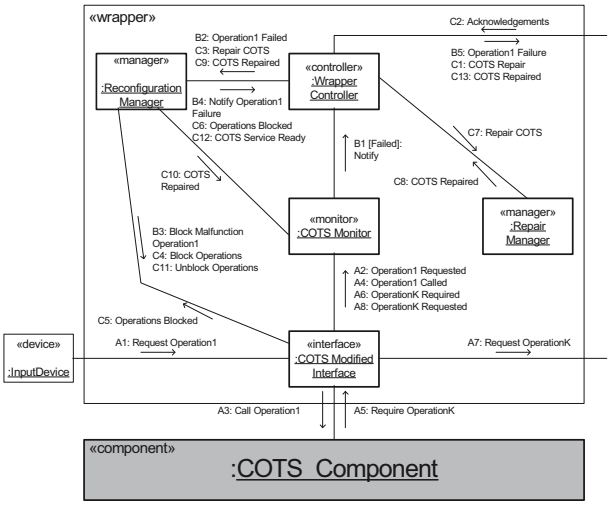


Fig. 1 Self-Management of COTS Component

The operations provided by a COTS component are monitored by the COTS Monitor. The COTS Modified Interface maintains the status of each operation in a COTS component. An operation is "unblocked" if it performs its obligation normally. An operation is marked "malfunction blocked" if the operation is anomalous, "dependency blocked" if the operation requires other operation in a different COTS component that is anomalous, or under repair block. Only unblocked operations are allowed to be invoked by the COTS Modified Interface. The message sequence A1 through A8 in the UML collaboration diagram [Booch05, Rumbaugh05] (Fig. 1) describes the monitoring of the Operation1 of the COTS component that is requested by an input device such as an elevator button.

The COTS Monitor presumes that an operation is anomalous if the expected notification messages have not arrived within reasonable time intervals (messages A2, A4, A6, and A8). The Wrapper Controller in a wrapper for a COTS component determines whether an operation in a COTS component should be repaired immediately or not - either just reconfiguring anomalous operations or reconfiguring/repairing the whole component – based on the criticality of operations.

The Reconfiguration Manager generates a reconfiguration plan against the anomalous operation. Based on the plan, it sends a message to the COTS Modified Interface to block the anomalous operation as "malfunction" (e.g., message B3). The Reconfiguration Manager also sends a failure notification to the neighboring COTS components (messages B4 and B5) through the Wrapper Controller.

An anomalous operation is repaired by the Repair Manager by means of re-initialization, re-installation, or replace of the component with a variant. In Fig. 1, the repair is performed by means of re-installation of the COTS component - message sequence C1 through C13.

3. Our Approach

By consideration of reusability, the wrapper for COTS components is designed and implemented into application-independent algorithms and application-dependent information. Application-independent algorithms for a wrapper are common to all COTS components so that they can be reused for different COTS components once they are implemented. Application-dependent information of a wrapper contains information specific to a COTS component, which should be updated for a different COTS component.

Fig. 2 depicts the Floor COTS Monitor, which is an object supporting the wrapper for Floor COTS component in the elevator system. The Floor COTS Monitor provides the following application-independent operations: Modified Interface Notification, Start Time, Monitor Time Notification, and Reinitialize Statechart.

Floor COTS Monitor
- Floor Operation Statechart Table - Floor Operation Statechart Information Table
- ModifiedInterfaceNotification(**in** operation, **in** event) - StartTime(**in** operation, **in** time) - MonitorTimeNotification(**in** operation) - ReinitializeStatechart(**in** operation)

Fig. 2 Floor COTS Monitor

Application-dependent information is described in a wrapper by means of tables. The Floor COTS Monitor (Fig. 2) has information specific to the Floor COTS component: a) Floor Operation Statechart Table (Table 1) that captures statecharts for the operations provided by the Floor COTS component; and b) Floor Operation Statechart Information Table that contains the time duration in which operations of the Floor COTS component should be performed.

Operation	State	Event	Action	Next Event
floorButton Request	Idle	floorButtonRequest Arrived	Start Timing	Calling Floor Button Request
floorButton Request	Calling Floor Button Request	floorButtonRequest Called	Start Timing	Processing Floor Button Request
floorButton Request	Processing Floor Button Request	serviceRequest Arrived	Start Timing	Calling Service Request
floorButton Request	Calling Service Request	serviceRequest Called	Start Timing	Idle

Table 1. Floor Operation Statechart

4. Self-Managed Elevator System

The elevator system with multiple elevators [Gomaa00] is structured into three Elevator COTS components, ten Floor COTS components, and one Scheduler COTS component. The objects in the wrappers for the COTS components were implemented using Java programming language. In the software architecture, each Elevator COTS component is allocated to an elevator, while each Floor COTS component is allocated to a floor. The Scheduler COTS component is allocated to a separate node for performance reasons, so it can rapidly respond to elevator requests.

4.1. Monitoring and Detection

The Floor Modified Interface has the Floor Operation Status table containing information about the status of Floor COTS operations: *floor button request*, *off floor lamp*, *on off direction lamp.* The status of each operation is unblocked so that the Floor Modified Interface can call the Floor COTS operations.

When the floor button sends an elevator request to the Floor Modified Interface, the Floor Modified Interface checks the status of *floor button request* operation in the Floor COTS component using the Floor Operation Status table. If the operation status is 'Unblock', the Floor Modified Interface notifies the Floor Monitor of the arrival of floor button request using the *Modified Interface Notification* function (Fig. 2). The statechart for the *floor button request* operation encapsulated in the Floor Monitor makes transition from the *Idle* state to the *Calling Floor Button Request* state (Table 1). The Floor Modified Interface then calls the *floor button request* operation in the Floor COTS component and notifies this event to the Floor Monitor by sending the "Floor Button

Request Called" message through the *Modified Interface Notification* function. The statechart makes transition from the *Calling Floor Button Request* state to the *Processing Floor Button Request* state (Table 1). Similarly, the Floor Modified Interface sends the next notification message, "Service Request Arrived", to the Floor Monitor when the Floor COTS component finishes processing the floor button request operation. By this message, the statechart makes transition from the *Processing Floor Button Request* state to the *Calling Service Request* state (Table 1). The Floor Modified Interface notifies the Floor Monitor again when it has sent "Service Request Called" message to the Scheduler Modified Interface. Then the Scheduler COTS component adds the floor and direction to the Scheduler's plan. This notification message makes transition from the *Calling Service Request* state to the *Idle* state in the statechart (Table 1).

Using the *Start Time* and *Monitor Time Notification* functions (Fig. 2), the Floor Monitor checks if each time interval between the notification messages from the Floor Modified Interface is within a specified time interval. For example, the Floor Monitor presumes that the *floor button request* operation is anomalous if the "Service Request Arrived" message is not arrived within a specified time interval. In this case, the Floor Monitor notifies the Floor Wrapper Controller of the anomaly of the operation.

4.2. Reconfiguration

The Floor Wrapper Controller has the Floor Operation Criticality table containing criticality information for each operation provided by the Floor COTS component. Operations are classified as a critical or non-critical operation based upon the criticality of system failure. The *floor button request* operation is critical for the elevator system because it handles the elevator requests from the elevator users. Thus, it needs to be both reconfigured and repaired immediately. The other two operations - *off floor lamp* and *on off direction lamp* - can be non-critical. Although the non-critical services provided by the Floor COTS component are degraded, malfunction of these operations may not affect the whole elevator system.

The Reconfiguration Manager has been implemented by means of the Reconfiguration Plan Generator and the Reconfiguration Plan Executor. The Reconfiguration Plan Generator generates a reconfiguration plan against an anomalous operation and the Reconfiguration Plan Executor performs the plan generated. To reconfigure the elevator system, the Floor Wrapper Controller sends the Floor Reconfiguration Plan Generator a message saying both the name of an anomalous operation and its level of criticality.

The Floor Reconfiguration Plan Generator checks the status of *floor button request* operation in the Floor/Callee/Caller Operation Status Table - containing the status of the operations provided by the Floor COTS component as well as their caller and callee operations - to determine whether the *floor button request* operation should be blocked or not. The Reconfiguration Plan Generator also uses the Floor Reconfiguration table (Table 2) to obtain the dependencies among operations. If there is some operation in caller components corresponding to the anomalous operation, the reconfiguration plan includes caller components so that the caller components should be notified for dependency block in response to the operation's anomaly. The *floor button request* operation will be blocked for repair (repair block) because the operation is critical. The *floor button request* operation has no operation in caller, so the plan does not include any operation of different components.

Operation	Callee		Caller	
	Operation	Component	Operation	Component
floorButtonRequest	Service Request	Scheduler		
offFloorLamp			upDownRequest	Elevator 1
			upDownRequest	Elevator 2
			upDownRequest	Elevator 3
onOffDirectionLamp			Approaching Requested Floor	Elevator 1
			closeDoor	
			Approaching Requested Floor	Elevator 2
			closeDoor	
			Approaching Requested Floor	Elevator 3
			closeDoor	

Table 2. Floor Reconfiguration Table

The Floor Reconfiguration Plan Executor performs all the actions in the reconfiguration plan. The Reconfiguration Plan Executor notifies the Floor Modified Interface to update the status of the *floor button request* operation in its table with "Repair Block". The Floor Reconfiguration Plan Executor will also change the status of the *floor button request* operation from "Unblock" to "Repair Block" in the Floor/Callee/Caller Operation Status Table. Then the Floor Reconfiguration Plan Executor notifies the Floor Wrapper Controller that the plan has been executed successfully. The Floor Wrapper Controller initiates the repair process by calling the Floor Repair Manager.

4.3 Repair

The Floor Repair Manager contains the Floor Repair Table, which shows different techniques that are available for repairing the anomalous COTS component. The repair techniques can be re-initialization of the COTS component, modification of inputs, re-installation of the same COTS component, replacement with a

variant, or any other technique that is available. The Floor COTS component has only two techniques available: re-installation and replacement.

The Floor Repair Manager starts repairing the Floor COTS component using the first technique that has not been used yet. If the technique selected is "Re-install component", a copy of the Floor COTS component kept in a safe place is re-installed. If the technique selected is "Replace component", a variant version of the Floor COTS component is selected as a new component. After executing one of those techniques, the Floor Repair Manager sends the "Component Repaired" message to the Floor Modified Interface to use this component version. The Floor Repair Manager notifies the Floor Wrapper Controller of the finalization of the repair.

When the Floor Wrapper Controller receives the notification from the Floor Repair Manager, it starts the reconfiguration process again as the Floor COTS component resumes its services. The Floor Wrapper Controller notifies the Floor Reconfiguration Plan Generator, which generates the corresponding reconfiguration plan again. The Floor Reconfiguration Plan Executor notifies the Floor Modified Interface to 'unblock' all of its own operations, and notifies the Floor Monitor to re-initialize the statechart for the *floor button request* operation using the *Reinitialize Statchart* function (Fig. 2). The Floor Reconfiguration Plan Executor changes the operations status in the Floor/Callee/Caller Operation Status Table to "Unblock", notifying the Floor Wrapper Controller that the repair process has been executed.

5. Conclusions

This paper has described a case study for self-managed COTS component-based elevator system in which each COTS component is encapsulated in a wrapper. While the wrapper for COTS components provides the properties of self-management, COTS components deal with application perspectives. By consideration of reusability, the wrapper is implemented into application-independent algorithms and application-dependent information.

This research can be extended to further research. The wrapper implemented needs to be applied to other applications to check how well the wrapper implemented is reused for other COTS components. In addition, the objects constituting a wrapper for COTS components may need to be refined to reduce the complexity of the wrapper architecture based upon the experience from case studies. The current implementation of a wrapper for COTS components requires many message communications between the objects in the wrapper, which result in low performance and resource overhead in the system.

References

[Anderson03] Tom Anderson, Mei Feng, Steve Riddle, Alexander Romanovsky, Protective Wrapper Development: A Case Study, in Proceedings of the 2nd International Conference on COTS-Based Software Systems (ICCBSS 2003), Ottawa, Canada, 10-13 February 2003.

[Booch05] G. Booch, J. Rumbaugh, I. Jacobson, "The Unified Modeling Language User Guide", Second Edition, Addison Wesley, Reading MA, 2005.

[Dashofy02] Eric M. Dashofy, Andre van der Hoek, and Richard N. Taylor, "Towards Architecture-based Self-Healing Systems," Workshop on Self-healing systems, Proceedings of the first workshop on Self-healing systems, Charleston, SC, November18-19, 2002.

[Garlan02] David Garlan and Bradley Schmerl, "Model-based Adaptation for Self-Healing Systems," Workshop on Self-healing systems, Proceedings of the first workshop on Self-healing systems, Charleston, SC, November18-19, 2002.

[Garlan03] David Garlan, Shang-Wen Cheng, and Bradley Schmerl, "Increasing System Dependability through Architecture-based Self-repair," in Architecting Dependable Systems. R. de Lemos, C. Gacek, A. Romanovsky (Eds), Springer-Verlag, 2003.

[Gomaa00] Hassan Gomaa, "Designing Concurrent, Distributed, and Real-Time Applications with UML," Addison-Wesley, 2000.

[Guerra04] Paulo Asterio de C. Guerra, Cecília Mary F. Rubira, Alexander Romanovsky, Rogério de Lemos, "A Dependable Architecture for COTS-Based Software Systems using Protective Wrappers," in Architecting Dependable Systems ADS II LNCS 3069. October 2004, pp. 147-170.

[IBM03] IBM, "An architectural blueprint for autonomic computing," IBM and autonomic computing, April 2003.

[Koopman03] Philip Koopman, "Elements of the Self-Healing System Problem Space," Workshop on Software Architectures for Dependable Systems (WADS2003), ICSE'03 International Conference on Software Engineering, Portland, Oregon, May 3-11, 2003.

[Meulen05] Meine van der Meulen, Steve Riddle, Lorenzo Strigini, Nigel Jefferson, Protective Wrapping of Off-the-Shelf Components, in Proceedings of the COTS-Based Software Systems: 4th International Conference, ICCBSS 2005, Bilbao, Spain, February 7-11, 2005.

[Rumbaugh05] J. Rumbaugh, G. Booch, I. Jacobson, "The Unified Modeling Language Reference Manual," Second Edition, Addison Wesley, Reading MA, 2005.

[Shin06] Michael E. Shin and Fernando Paniagua, "Self-Management of COTS Component-Based Systems Using Wrappers," 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), Chicago, September 17-21, 2006.

[Shin07] Michael E. Shin and Fernando Paniagua, "Design of Wrapper for Self-Management of COTS Components," 19th International Conference on Software Engineering and Knowledge Engineering (SEKE'2007), Boston, July 9-11, 2007, pp. 314-319.

[Thomas00] V. Thomas, SuriKumar Kareti, Walter Heimerdinger, Sunondo Ghosh, "Mediators and obligations: An architecture for building dependable systems containing COTS software components," Proceedings Workshop on Dependable System Middleware and Group Communication (DSMGC 2000), Nuremberg, Germany, October 2000.

Using Scenario Monitoring to Address State Based Crosscutting Concerns

Mark Mahoney[1, 2]
Tzilla Elrad[2]
[1]*Carthage College*
[2]*Illinois Institute of Technology*
mmahoney@carthage.edu
elrad@iit.edu

Abstract

This paper describes how the completion of well defined scenarios can be used to indicate events of interest to state based crosscutting concerns. Core concerns are monitored for scenarios that represent events of interest to a crosscutting concern. When a monitored scenario completes, an event is injected into a crosscutting state machine that may react by introducing additional behavior. This is a form of Aspect-Orientation that deals with non-state based core concerns interacting with state based crosscutting concerns.

1. Introduction

A crosscutting concern is an aspect of a system that influences many other core concerns. Crosscutting concerns cannot be easily modularized using traditional decomposition techniques. Fault tolerance, for example, is a crosscutting concern that affects many parts of a system. However, fault tolerance code is typically scattered throughout the system and tangled with the core concerns interfering with their logical flow. The field of Aspect-Oriented Software Development (AOSD) [1] addresses crosscutting concerns by separating them from core concerns at one level of abstraction and providing a means to weave them back together at a lower level of abstraction. The woven product is one step closer to an executable system. For humans analyzing the system, the separation of concerns allows one to reason about core and crosscutting concerns independently while understanding how they affect each other.

Many core concerns exist that have no state based behavior. They do not require any knowledge of the past in order to satisfy a requirement. Occasionally, a crosscutting concern requires knowledge of a core concern's history or state in order to function properly. Consider a banking system with different types of accounts that are accessible from a bank teller, ATM, or online. From a security standpoint, repeated transfers in a single day through an ATM or online rather than through a bank teller might require that the transaction be logged as suspicious activity. The core transfer behavior is not state based and does not require any state information to function properly. The crosscutting security logging concern, however, does require knowledge of the core's state. In particular, it needs to know how many transfers have been attempted in a day and by what means the transfers took place. This is an example where tracking the state of a core concern is necessary for a crosscutting concern to behave correctly.

A state machine can be used by a crosscutting concern to model the history information of the core, but it should not be tightly coupled to any particular core concern. It should be abstract so that the crosscutting concern is usable in different contexts and by many different core concerns. The events that this crosscutting state machine reacts to must come from the core concerns. This is how core and crosscutting concerns are woven together. The obliviousness property of AOSD [6] states that the core concerns need not be aware of any crosscutting concerns affecting them. The core developer must not be responsible for creating a state machine if it is used solely for a crosscutting concern.

The contribution of this work is to show the benefits of monitoring a set of non-state based core concern objects for events that are useful to a state based crosscutting concern. In the example above, the core objects can be monitored to determine when a user transfers money using an ATM or online. When that scenario occurs it can be used as an event in a state machine for the security logging crosscutting concern. This provides separation of concerns and maximizes reusability of both core and crosscutting concerns.

In order to perform the monitoring several approaches can be used. The least invasive approach uses combination of well known patterns to track the state of a set of core concern objects. However, other tools and language extensions exist that can be used for the same purpose.

The rest of this paper is organized as follows: section two briefly discusses different approaches to scenario monitoring and gives an overview of our approach. Section three provides an example of using scenario monitoring to inject events into a crosscutting concern state machine. Section four describes related work.

2. Scenario monitoring for generating events

In our approach, a crosscutting concern developer models the state of the core concerns with a state machine. That is, the crosscutting concern has a particular need to know the state of the core concerns. This is so that behavior can be injected on certain transitions of the core's state. The crosscutting state machine is abstract in that the events are not directly bound to a core concern's implementation. Rather, the core concerns are monitored for events of interest. A scenario is an ordered set of messages sent and received from objects in the system. Traditionally, scenarios are modeled with sequence diagrams [18].

When the core objects are monitored and an event of interest occurs the event will be sent to a state machine that may react to it. This is how additional behavior is added to the core concerns. Scenario monitoring is the crucial element in this approach. Two different approaches to scenario monitoring will be discussed in the following subsections.

2.1. Scenario Monitor Pattern

Ideally, scenario monitoring should be done in a minimally invasive way. Requiring the monitored objects to be aware of the monitoring introduces strong coupling between the core concerns and the monitoring code. The coupling makes it difficult to reuse the core concerns in a context where scenario monitoring is not required.

In an upcoming work we will present a design pattern that allows monitored objects to be oblivious to monitoring code. Scenario monitoring can be turned on and off at run time. Although the exact details are beyond the scope of this work, the pattern makes use of the Decorator, Observer, and Abstract Factory Patterns [7] to shield the core concern developer from being aware that monitoring is taking place.

2.2. The Play-Engine

Harel et. al. [9][15][8] created Live Sequence Charts to specify scenarios and reactions that occur in response to the completion of those scenarios. In this approach a proprietary tool called the Play-Engine monitors scenarios. In Maoz [15] the idea was reworked to eliminate a separate monitoring tool. Instead, monitoring is accomplished using the Aspect Oriented Programming language AspectJ [11]. The only drawback of these approaches is that they cannot be used without committing to new tools and programming languages.

2.3. Overview of our approach

Although we prefer scenario monitoring solutions that don't require special language extensions or tools, any of the approaches above can be used to separate non-state based core and state based crosscutting concerns. State based crosscutting concerns are modeled with state machines. The state machines have states, transitions, and actions associated with transitions. These actions are used to weave in additional behavior into a system. The state machines, however, should not be coupled directly to any particular set of core concerns. The state machine should receive abstract events. A mapping must be made from the concrete core concerns to the abstract events that will be handled by the crosscutting state machine.

Our initial approach is for completed scenarios in the core concerns to represent events of interest. This is reasonable because an event is an occurrence in time and space that has significance to the system [16]. Therefore, the scenario monitor will inject events into the crosscutting state machine and it will react accordingly, perhaps changing state and executing behavior associated with the transition.

We feel an example of our approach is the best way to explain our approach. The following section gives a detailed example on a relatively small set of requirements.

3. Example using our approach

The following describes a set of requirements that we use to elucidate our approach. The system is for a financial advisor that generates and sells reports to his customers about potential companies to invest in. The financial advisor gets some of his financial data from a much larger financial services organization referred to as the Investment Warehouse.

3.1. Report generating system

Requirement R1: Financial Advisor Attempts to Sell Report

A financial advisor requests an investment report from the Report Generating System to sell to his customer. If the Report Generating System does not have the requested report it will ask an independent Investment Warehouse for information in order to generate the report. The Report Generating System will then generate a report and send a summary to the financial advisor. The financial advisor shows the customer the summary and tries to sell it to her. If the customer wishes she may purchase the full report. If that happens the report will be stored by the system and presented to the customer. When a customer requests a report that already exists, the Report Generating System will pull the report from storage and display a summary to the customer. If the customer chooses to purchase the report it will be presented to her.

The first requirement is modeled with the following use case:

Use Case: Financial Advisor Attempts to Sell Report
Actors: Financial Advisor, Investment Warehouse
Normal Flow:

1. Financial Advisor requests a report from the System.
2. System searches report database for existing report.
3. The report does not exist, System requests information from Investment Warehouse.
4. System generates the report.
5. System sends summary report to the Financial Advisor.
6. Financial Advisor gets approval from the customer to purchase the report.
7. System stores the report in the database.
8. Financial Advisor presents the report to customer.

Alternate Flow: Report Already Exists

3. The report already exists, retrieve it from the database.
4. System sends summary report to the Financial Advisor.
5. Financial Advisor gets approval from the customer to purchase the report.
6. Financial Advisor presents the report to customer.

In the early design phase the Sequence Diagrams in figures 1 and 2 are created.

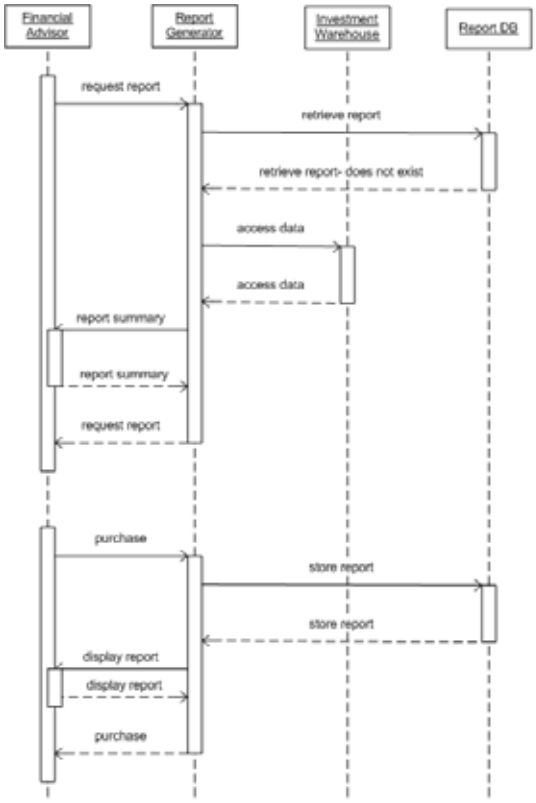


Figure 1. Financial advisor attempts to sell report (new report)

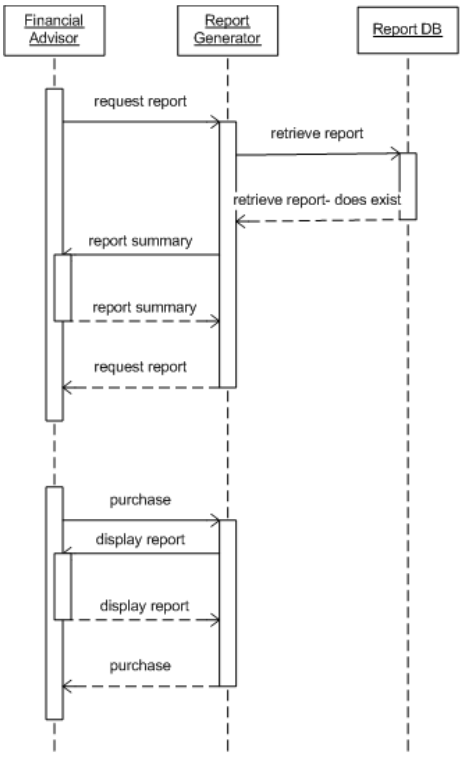


Figure 2. Financial advisor attempts to sell report (report already exists)

3.2. Crosscutting billing concern

Now imagine the financial advisor has the option of becoming a franchisee of the Investment Warehouse or a pay-as-you-go customer. If the financial advisor becomes a franchisee, he is charged a relatively high flat yearly rate for access to financial services information. A pay-as-you-go customer is charged per access to the Investment Warehouse. The financial advisor has negotiated an additional term in the contract. The Investment Warehouse will only charge the financial advisor once when he accesses data the first time a report is sold. If the financial advisor does not sell the report, or the report is already in his database of sold reports, he is not charged for the data access.

The financial advisor would like the flexibility to switch between the different types of customer. If he switches he would like to keep his existing Report Generating System in place. The main difference is that a franchisee does not need to handle billing.

The state machine in figure 3 describes how pay-as-you-go customers are charged for access to the Investment Warehouse's data. The requirement is to charge $10 for the first 5000 accesses in a month, $5 for the next 5000 accesses in a month, and charge nothing for more than 10000 accesses in a month.

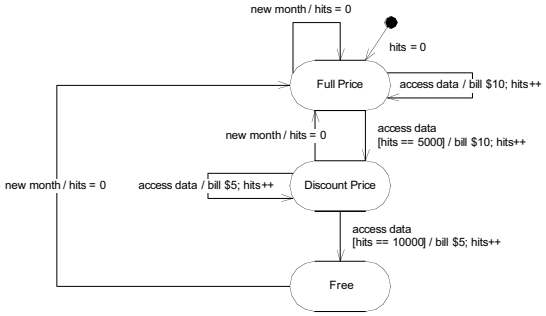


Figure 3. 'Access Data' state machine

The problem is determining when a billable 'access data' event has been performed. From the requirements it is clear that the only time a billable access occurs is when a new report is actually purchased. One cannot simply bill after accessing the data from the Investment Warehouse because there is no charge unless the customer purchases a report. Further, one cannot simply bill after the customer purchases a report because they may be purchasing a report that already exists. The billing system needs to know the state of the interaction with the Investment Warehouse.

The important states to the billing concern are Idle, Pending Purchase New Report, Pending Purchase Existing Report, and New Report Purchased. The important events are when a new report is generated, when an existing report is requested, when a report is purchased, and when a report is abandoned. The state machine in figure 4 describes when billing should take place.

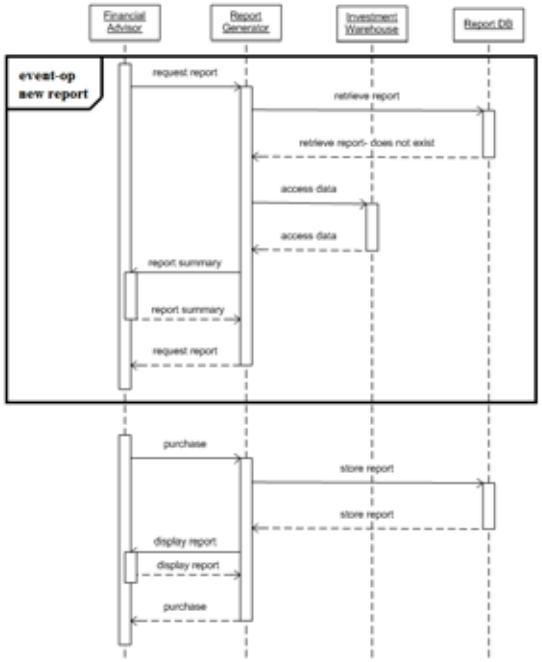
Figure 4. 'Billing' State Machine.

3.3. Concern Modeling

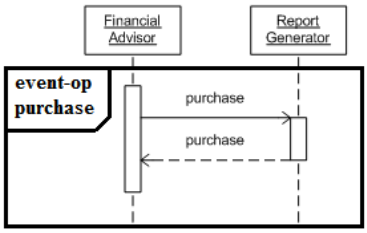
Scenarios of interest can be modeled in a variation of UML interaction diagrams. We are proposing the use of Interaction Fragments from UML 2.0 to specify scenarios that will map to events in a crosscutting

concern state machine. Interaction Fragments model a sequence of events that has special properties to a developer [18]. A Combined Fragment is a container of Interaction Fragments with operators specified for the contained Interaction Fragment. We are proposing a new interaction operator called 'event-op'. The 'event-op' operator, when applied to Interaction Fragments, specifies that upon completion of the Interaction Fragment a corresponding event or events will be introduced into one or more state machines. The state machine may react to the events by introducing behavior associated with the transition.

The state machine in figure 4 describes (in an abstract way) when the crosscutting billing concern should be applied. What remains is the concrete mapping of scenarios in the core objects to the events in the crosscutting concern state machine. The events 'new report', 'report exists', and 'purchase' are directly related to the completion of certain scenarios. The 'new report' event from the 'Billing' state machine is bound to the completion of the scenario specified in figure 5a. That is, the completion of that part of the sequence diagram is equivalent to the event where a new report is created. Whether that event is handled is determined by the current state of the crosscutting concern state machine. Similar events are bound to the scenarios 'purchase' and 'report exists' in figure 5b and 5c.



a. New Report Event



b. Purchase Event

c. Report Exists Event

Figure 5. Specification of bindings between scenarios and events.

4. Related Work

There is some existing research that has addressed scenario monitoring and AOSD. Trace-Based Aspects [5], also referred to as Stateful Aspects, are defined on traces of events that occur during program execution. Stateful Aspects use system state to represent the evolution of crosscutting concerns according to events that are encountered. Stateful aspects have two main characteristics. First, aspects are defined over sequences of observable execution states. Second, weaving is performed on executions rather than program code. The weaver can be seen as a monitor interleaving the execution of the base program and execution of inserts.

The Aspect-Oriented Programming language JASCO [10] was extended to allow for the implementation of Stateful Aspects. The additional constructs permit the order of messages to be monitored and behavior to be applied upon completion of scenarios.

In [2] the authors argue that Stateful Aspects are a consequence of designing a state based system using transformational techniques. They propose an approach to dealing with crosscutting concerns in state based systems using state based design techniques and tools including the Specification and Design Language (SDL)[3][4]. We believe there is a place for Stateful Aspects in transformational systems if they can be modeled correctly.

The work of Stein [17] relates to modeling. In that approach, one models the state of the core concerns with state machines like we do but the events come directly from individual messages from the core concern models. That is, the events in the state machines are directly bound to method invocations from the core. We provide a more abstract model that allows one to specify the concrete messages outside of the crosscutting concern. The events in the state machine using our approach are not coupled to the messages in the core implementation. Using Stein's approach it would be difficult to use the crosscutting state machines in other implementations because they are coupled to particular core models.

5. Conclusion

History sensitive crosscutting concerns are difficult to implement when the history lives in the core implementation. We have described a way to monitor the core concerns and create events for the crosscutting concerns.

The primary benefit of our approach is the loose coupling between core and crosscutting concerns. The specification of binding between the core and crosscutting concerns is at a higher level of abstraction than other approaches. The consequences are that developers can specify state based behaviors required for crosscutting concerns in an abstract way that is reusable in different contexts. Crosscutting concern developers can emphasize the state based nature of concerns without requiring the core concern developers to create a state machine model- they are oblivious.

6. References

[1] AOSD web page. http://aosd.net

[2] Cottenier, T. van den Berg, A. Elrad, T. Stateful Aspects: The Case for Aspect-Oriented Modeling. Proceedings of AOM @ AOSD'07 10th Int'l Workshop on Aspect-Oriented Modeling Vancouver, Canada, 12 March 2007

[3] Cottenier, T., van den Berg, A., Elrad, T. Joinpoint Inference from Behavioral Specification to Implementation, ECOOP'07

[4] Cottenier, T., van den Berg, A., Elrad, T. The Motorola WEAVR: Model Weaving in a Large Industrial Context. Industry Track paper at AOSD'07

[5] R. Douence, P. Fradet, M. Sudholt Trace-based AOP in M. Aksit, S. Clarke, T. Elrad, R. E. Filman editors, Aspect oriented software development, Addison-Wesley, 2004

[6] Filman R.E., Friedman, D.P. "Aspect-Oriented Programming is Quantification and Obliviousness", Workshop on Advanced Separation of Concerns, OOPSLA 2000, October 2000, Minneapolis.

[7] Gamma, Helm, Johnson, Vlissides; Design Patterns, Elements of Reusable Software Design, Addison-Wesley 1995

[8] D. Harel and S. Maoz, "Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams", Software and System Modeling (SoSyM), to appear. (Early version in 5th Int. Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM'06), 2006.)

[9] Harel, D., Marelly, R. Specifying and Executing Behavioral Requirements: The Play In/Play-Out Approach Software and System Modeling (SoSyM) 2 (2003), pp. 82-107.

[10] JASCO Home Page. http://ssel.vub.ac.be/jasco/

[11] Kiczales, G. et al., Aspect-Oriented Programming Proc. European Conf. Object-Oriented Programming, Lecture Notes in Computer Science, no. 1241, Springer-Verlag, Berlin, June 1997, pp. 220-242.

[12] Mark Mahoney, Atef Bader, Tzilla Elrad, Omar Aldawud, Using Aspects to Abstract and Modularize Statecharts, The 5th Aspect-Oriented Modeling Workshop in Conjunction with UML 2004 Lisbon, Portugal, October 2004

[13] Mahoney, M., Elrad, T. A Pattern Based Approach to Aspect Orientation for State Based Systems, Workshop on Best Practices in Applying Aspect Oriented Software Development (BPAOSD ' 07) at the Sixth International Conference on Aspect-Oriented Software Development (AOSD 2007). March 2007. Vancouver, BC.

[14] Mark Mahoney, Tzilla Elrad, Weaving Crosscutting Concerns into Live Sequence Charts Using the Play Engine. 7th International Workshop on Aspect-Oriented Modeling held in conjunction with the 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS'05), Montego Bay, Jamaica, October 2005.

[15] Maoz, S. Harel, D. From multi-modal scenarios to code: compiling LSCs into aspectJ Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering Portland, Oregon, USA 2006: 219 – 230

[16] Samek, M. Practical Statecharts in C/C++. CMP Books. 2002.

[17] Dominik Stein, Stefan Hanenberg, Rainer Unland: Visualizing Join Point Selections Using Interaction-Based vs. State based Notations Exemplified With Help of Business Rules. EMISA 2005: 94-107

[18] UML Specification. http://www.uml.org/

Negotiating Service Levels - A generic negotiation framework for WS Agreement

Sebastian Hudert
Chair of Information Systems Management, University of Bayreuth
95440, Bayreuth, Germany
Email: sebastian.hudert@uni-bayreuth.de
Heiko Ludwig
IBM Research, T.J. Watson Research Center
Hawthorne, New York, USA
Guido Wirtz
Distributed and Mobile Systems Group, University of Bamberg
96045, Bamberg, Germany

Abstract

The Web Services Agreement specification draft proposes a simple request-response protocol for agreement creation only addressing bilateral offer exchanges. Analyzing negotiation taxonomies from the literature clearly proves that as insufficient. This paper proposes a framework that augments the WS-Agreement in order to enable negotiations according to a variety of bilateral and multilateral negotiation protocols in a manner that is compatible with the WS agreement standard.
Keywords: SLA, Negotiation Protocols, WS Agreement

1 Introduction

Managing quality of service (QoS) in loosely-coupled distributed systems such as computational Grids cannot rely on traditional, centralized management. QoS guarantees must be obtained in the form of bi- or even multi-lateral service level agreements (SLAs). SLAs represent qualitative guarantees placed on service invocations within a service oriented environment. Service consumers benefit from guarantees because they make non-functional properties of service predictable. On the other hand, SLAs enable service providers to manage their capacity, knowing the expected requirements. By employing SLAs, a robust service oriented architecture (SOA) can be realized, even across company boundaries.

To support broad application, standards for the structure of agreement documents as well as a standard process to establish and monitor them automatically are required. The Web Services Agreement (WS-Agreement) specification is a standardization effort conducted by the Open Grid Forum (OGF) in order to facilitate creation and monitoring of SLAs [1]. It defines an XML-based structural definition of SLA documents, a request-response protocol for agreement creation as well as corresponding interfaces for agreement creation and monitoring. A WS-Agreement specifies functional properties and qualitative service level guarantees by a set of terms.

Unfortunately, the agreement creation process proposed by the standard is restricted to a simple request-response protocol. Taking a closer look at the literature on negotiation taxonomies originating in e-commerce research and economics shows that simple request-response pattern are far from being sufficient to constitute complex SOAs. Enabling a variety of negotiation protocols supporting advanced formats involving numerous parties in different roles would result in a much wider applicability of the WS-Agreement standard.

In order to have a common basis for a more powerful framework for automated negotiation, a thorough study of the literature has been carried out (refer to [7] for details). Whereas much work is focussed on auctions ([3, 14, 15]), recent work originating in electronic negotiation research particularly deals with e-commerce settings ([8, 2]). The detailed taxonomy by Ströbel and Weinhardt [13] provides in our opinion a classification of negotiation parameters independent of concrete technologies. The requirements and characteristics discussed in the literature have been used in an integrated and consolidated manner to derive a set of attributes that are suitable for SLA negotiation settings supporting various auction types or one-on-one bargaining protocols.

Current SLA negotiation frameworks lack the ability to provide different negotiation protocols. Such efforts either only support one particular negotiation protocol that is fixed within the system (see for example [4, 5]) or even employ hierarchical management structures that hardly negotiate with the resource providers at all but act as brokers distributing jobs on the registered services (see [6], [11]). Other systems provide the ability of different protocols but only by introducing a central market instance mediating all negotiations ([10]), which in our view contradicts the decentral nature of distributed service-based systems.Our framework aims at overcoming this lack of generity in SLA negotiation mechanisms as proposed up until now.

Incorporating different negotiation protocols seamlessly into the agreement creation process of the overall WS-

Agreement protocol, however, requires solving several issues: In an automated negotiation, all participating components (called *agents*) must be aware of all rules and constraints concerning the negotiation protocol. Moreover, an infrastructure of role definitions, interfaces and methods has to be presented to facilitate the actual negotiations.

Although publishing a fixed, predefined set of negotiation protocol definitions would suffice for lots of scenarios, faced with the rapid development of SOA and its applications, a more generic approach is more appropriate. We propose a *generic approach* in which parties in a distributed system agree on a negotiation mechanism first, then conduct the SLA negotiation and then fulfill the SLA. To this end, we define a *meta-language for negotiation protocols*. Using such a meta-language, a multitude of specific negotiation protocols can be defined using a well-defined set of attributes and parameters. Making a specific protocol definition available to all prospective negotiators before the actual negotiation informs them about which protocol has been chosen. Further, we propose an *exchange protocol* to distribute the negotiation definitions to all prospective negotiators and to choose a specific negotiation protocol. Finally we propose, as an example, a *generic negotiation protocol* that is able to support all specific negotiation protocols that can be described with the presented negotiation attributes as extension to basic WS-Agreement offers.

The rest of this paper is organized as follows: Section 2 introduces the basic concepts and data structures of the framework. Afterwards, in sections 3 and 4, the underlying philosophy as well as the interfaces and building blocks offered for the exchange protocol and the negotiation protocol itself are described. We conclude with some remarks on future work.

2 Basic Definitions and Data Structures

The negotiation protocol actually conducted is defined with the means of a negotiation meta-language. In this section we sketch the meta-language and data structures forming the basis for our framework. A multitude of different negotiation protocols can be specified using the provided well-defined set of negotiation parameters, resulting in a structured protocol description document. There are four general negotiation configurations:

1:1 exactly two negotiators exchange bids, i.e. *Bargaining*

1:n classical auctions, where n service consumers post bids to a single service provider

n:1 reverse auctions where n service providers post offers to a single service consumer

n:m (broker) markets, where n service providers and m service consumers post bids to a central broker that matches offer and demand

Our framework currently covers bargaining situations, and (reverse) auctions. Incorporating markets will be subject to future work. Based on the literature on negotiation taxonomies (cf. section 1), a set of *negotiation attributes* has been identified by analyzing common SLA scenarios which inherently exhibit distinct characteristics:

- SLAs normally comprise more than just a single attribute, therefore an SLA framework must focus on combinatorial negotiations (in contrast to, e.g., [14])
- Each service is referenced individually and defines an individual item. Hence multi-unit negotiations are not appropriate in SLA negotiation settings.
- An SLA always governs one or more service invocations done by a service consumer. The service is offered by the service provider. Therefore SLA negotiations primarily focus on these two roles in a negotiation.
- In order to guarantee integrity of the negotiation a common concept in negotiation theory is a trusted third party governing the negotiation process. This is also appropriate for SLA environments in which service consumers and providers can utilize such a central service for discovery of potential negotiation partners.
- A special requirement posed here is the need for *fully automated negotiations*. Negotiation protocol descriptions used in such scenarios must conform to a very strict structure and must be syntactically processable by software agents negotiating on service consumers'/providers' behalf.

A detailed description of our taxonomy can be found in [7] along with a data model for negotiation protocol descriptions, formalized as an ER Model, as well as an XML- based representation of this data model for seamless integration with the WS-Agreement specification. Based on the analysis the following high level attribute categories were identified:

- *General Negotiation Process* attributes abstractly define the overall negotiation process, e.g., the starting and termination rules for a negotiation, the number of rounds or whether or not the negotiation protocol is rewarding protocol compliance or punishing protocol violation, i.e. employing reputation concepts.
- *Negotiation Context* groups attributes define a negotiation's configuration. This includes for example the definition of roles and the admission rules for each role.
- *Negotiated Issues* attributes define the values of the SLA to be negotiated and, hence, which attributes of a service are subject to the negotiation and to which extend this set can be extended.
- *Offer Submission* attributes govern the bidding process. Rules concerning the submission of bids or the relation between bids are specified with these attributes. They define which roles are allowed to post bids and under which conditions a bid can be posted or is evaluated to be valid within the negotiation.
- *Offer Allocation* attributes govern the matching process of a negotiation, more precisely the agreement formation in SLA scenarios.
- *Information Processing* attributes determine what kind of information, e.g., about the current status of the negotiation or past offers from negotiating agents, are accessible during a negotiation.

Employing the identified attributes, an XML Schema document describing the structure of our negotiation protocol description documents, so-called *negotiation types*, are derived.

2.1 Negotiation Types and Instances

In order to allow for the specification of a reusable protocol definition we distinguish two different protocol description documents: negotiation types and instances. Negotiation types describe general classes of negotiation protocols and define their common attributes and elements. Negotiation instances, in contrast, stand for a particular negotiation of some type that can be unambiguously referenced by (potential) participants. A negotiation type document can therefore be seen as an instantiated protocol description employing our meta-language, whereas a negotiation instance refers to a particular negotiation process conducted according to such a protocol description. A negotiation type does not contain a identifier that is used to refer to a given negotiation process and does not state the agents involved in a given negotiation, since they change between each actual negotiation process. The other attributes identified in the previous section, however, have to be initialized when defining a negotiation type as the negotiation type essentially describes the protocol to be executed.

The involved agents are not specified in a negotiation type document because not every agent participating in the negotiation should have to be known when a negotiation type is created. Such a constraint would corrupt the flexible approach of our framework. Instead, the exchange protocol (see section 3) allows for agents to subsequently join the negotiation. Hence the negotiating agents will subsequently filled in the negotiation instance document.

In order to supply the negotiating agents with the required information about negotiation types and instances, two XML document descriptions (formalized XML-Schema documents) were defined in a manner that is suitable to exhaustively describe a multitude of different 1:1, n:1 and 1:n negotiation protocols ([7]).

2.2 Negotiation Documents

The main negotiation object is a WS-Agreement template [9] with its corresponding creation constraints as defined in the current WS-Agreement specification. The framework augments the current specification with possibilities to negotiate over a WS-Agreement by adapting this fundamental data structure for the (partial) definition of some service(s) to be negotiated. The creation constraints as part of this template are also used to give syntactical restrictions on the elements still to be initialized or to be altered during the negotiation. The *negotiation type document* refers to the WS-Agreement template the negotiation is defined upon and defines which terms of a WS-Agreement can be negotiated and how to do so. A concrete negotiation is represented by a *negotiation instance document* as already hinted. This document refers to the negotiation's type, its participants and specifies a unique identifier. Finally, the result of the complete negotiation protocol is a valid WS-Agreement document satisfying the initial creation constraints of the WS Agreement template it is based upon.

Figure 1. Agreement Creation Process

2.3 Involved Roles

Since our framework is intended to augment the WS-Agreement specification, we identified a set of service interfaces representing the roles present in a SLA negotiation process. Each of these roles offers a distinct functionality to the other actors in the system in order to enable the exchange and negotiation protocols (see sections 3, 4). The *Negotiation Participants* represent a regular participant/negotiator in the exchange and negotiation protocol. In terms of service oriented environments the service consumers and providers make up such negotiation participants. The *Negotiation Coordinator* is a logically centralized instance which handles admission of agents to a given negotiation as well as (re)distribution of the negotiation documents to the prospective negotiators. The information distribution during the actual negotiation is administrated by the *Information Service*. It serves as an access point for the negotiators to access information about the current negotiation process.

3 Exchange Protocol

The overall process of agreement creation can be divided into four distinct phases, as depicted in Figure 1: First an appropriate protocol description in terms of our meta-language and the respective negotiation type and instance documents is created. In a second step this protocol description is distributed to all prospective negotiators according to the *exchange protocol* presented here. Subsequently the actual negotiation process takes place according to the rules defined and distributed in the previous phase. The generic negotiation protocol used for this phase is presented in the next section. Finally one offered agreement is accepted by one of the participants to conclude the negotiation. Alternatively, there may be no acceptable offer and the negotiation is terminated by rejecting all offers. Neither the exchange nor the negotiation protocol definition focus on exchanged messages primarily, but on the provided services and respective methods to be invoked subsequently. This approach was taken due to the envisioned service oriented environment for our framework defining protocols in terms of method invocation sequences.

3.1 Interfaces for the Roles Involved

During the exchange process a negotiation instance document is distributed to all prospective negotiators. At the end of the exchange protocol every involved agent should be aware of the negotiation protocol to be executed and all agents involved as negotiators. Hence two roles are present in the exchange process: the *Negotiation Participants* joining a given negotia-

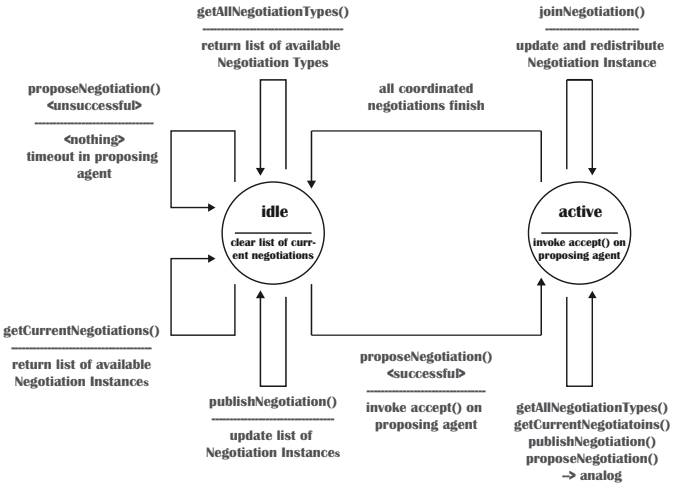


Figure 2. NegotiationCoordinator statemachine

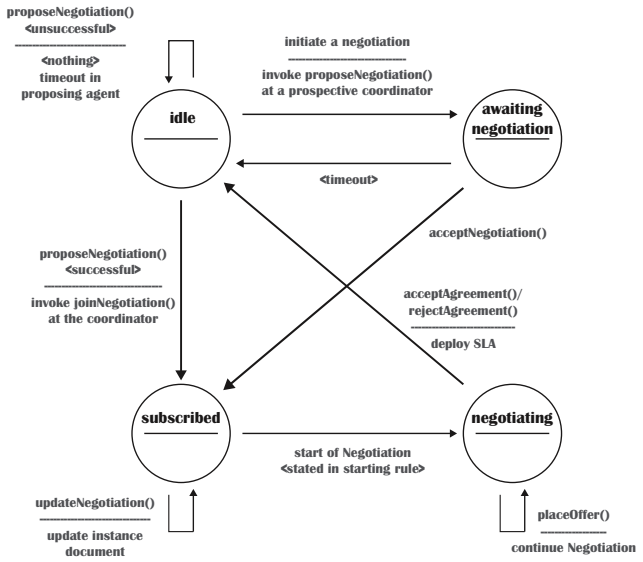


Figure 3. NegotiationParticipant statemachine

tion and a centralized *Negotiation Coordinator* governing the distribution of the protocol description documents.

The Negotiation Coordinator provides the protocol description documents and handles the admission of negotiators. The corresponding interface defines a set of query methods used for requesting the available negotiation type and instance documents.

- *getAllNegotiationTypes()* / . . .*TypesForTemplate(. . .)*
- *getCurrentNegotiations()* / . . .*ForTemplate(. . .)*

Employing these methods the negotiators can query all available negotiation types as well as currently active negotiation instances. Additionally it is possible to use a given WS-Agreement template as a search parameter in order to query all possible negotiation protocols for a given service. Besides simply joining an already running negotiation, an agent may actively propose a negotiation instance to a coordinating agent.

- *joinNegotiation(negotiationID, agentEPR, 'credentials')*
- *proposeNegotiation(NegotiationInstance-document)*
- *publishNegotiation(NegotiationInstance-document)*
- *publishNegotiationToReceipients(. . ., [receipients])*

Processing admission of agents at one logical centralized coordinator service eases the integration of reputation or security related external systems involved in the admission process. All agents joining a negotiation do so by invoking the corresponding method on the central coordinator service.

There are two different types of methods proposing a negotiation instance to the coordinator: proposeNegotiation() and publishNegotiation(). The latter differs from the former in that it is not assumed that the coordinator used for publishing also is to act as Negotiation Coordinator for the respective negotiation. It only offers this negotiation instance for look-up purposes while the actual admission and information (re)distribution tasks are conducted by the actual coordinating agent, probably the one publishing the negotiation instance. This method can be used to implement systems of distributed look-up servers. With the *proposeNegotiation()*-method a negotiation instance is proposed to the agent acting as Negotiation Coordinator in the subsequent negotiation process. Figure 2 outlines the overall behavior when using the *NegotiationCoordinator* interface.

The Negotiation Participant role is adopted by all prospective negotiators, i.e., by service providers as well as consumers. However, this role is present in both, the exchange and the negotiation protocol. The methods used for the exchange protocol are described here while the ones used in the actual negotiation will be sketched in the context of section 4.

- *updateNegotiation(NegotiationInstance-document)*
- *proposeNegotiation(NegotiationInstance-document)*
- *acceptNegotiation(negotiationID)*

Whenever new agents join a negotiation, the negotiation instance document is updated and redistributed to all negotiators already known using the *updateNegotiation()*-method. On the other hand, it should be possible for a Negotiation Coordinator to propose a negotiation instance to a (potential) Negotiation Participant. For this purpose a *proposeNegotiation()*-method is also present in the Negotiation Participant interface. The *acceptNegotiation()*-method is offered as a counterpart for the *proposeNegotiation*-method to support asynchronous communication. When a negotiation is proposed to a Negotiation Coordinator, this agent can decide whether to coordinate this negotiation or not using the *acceptNegotiation()*-method on the proposing agent. Figure 3 sketches the overall behavior of a *NegotiationParticipant*).

Based on the two roles explained above, our framework offers three basic logical protocol components that may be combined in order to create a concrete exchange process.

3.2 Request for Negotiation Documents

This step describes the process of one agent requesting negotiation type or instance documents from the respective Nego-

tiation Coordinator, employing the methods defined in section 3.1.

An agent may, for example, query all negotiation types supported by the respective Negotiation Coordinator using the *getAllNegotiationTypes()*-method. As a result the coordinator returns a list of negotiation type documents. On the other hand, agents may query already instantiated negotiations with the *getCurrentNegotiations()*. In the former case the Negotiation Coordinator only defines which protocols are supported and waits for the other agents to propose a particular type to be instantiated. In the latter the coordinator returns all currently available instances for the requesting agent to join. Analogously to requesting all available negotiation types or instances, agents may also query only types and instances defined for a given WS-Agreement template, i.e. for some service they already know. If one of the returned negotiation types or instances is appealing for the requesting agent and it wishes to take part in the respective negotiation, the involved agents have to conduct an additional step. In case of the request for negotiation types an agent can create a corresponding negotiation instance document and propose it to the coordinator (see next subsection). The proposing agent can join a successfully instantiated negotiation instance by invoking the *joinNegotiation()*-method afterwards.

If an agent wants to join an already instantiated negotiation, the agent requests the currently available negotiation instances first, chooses an appropriate one and invokes the *joinNegotiation()*-method on the coordinator afterwards.

3.3 Proposal of Negotiation Documents

This step represents the process of actively proposing some instance document to a prospective participant or coordinator. This way negotiations can either be proposed to agents simply taking part in or to some agent coordinating the subsequent bidding process. The protocol component regularly follows a request for negotiation types in order to propose the newly created instance to the coordinating agent.

3.4 Mediated Exchange Processes

The third building block offers publish/subscribe functionality to the participants. Agents may publish negotiation instances at some Negotiation Coordinator to make it available to a larger community of prospective negotiation participants.

In order to implement such systems of distributed look-up servers, the Negotiation Coordinator offers the *publishNegotiation()* interface method. This method allows for publication of instantiated negotiations at some coordinating service. The other agents requesting the available protocols again query these by invoking the already introduced request-methods.

By combining these three basic protocol components as building blocks, a multitude of different exchange processes can be specified, all resulting in distributing the information, needed to participate in a particular negotiation, to all prospective participants.

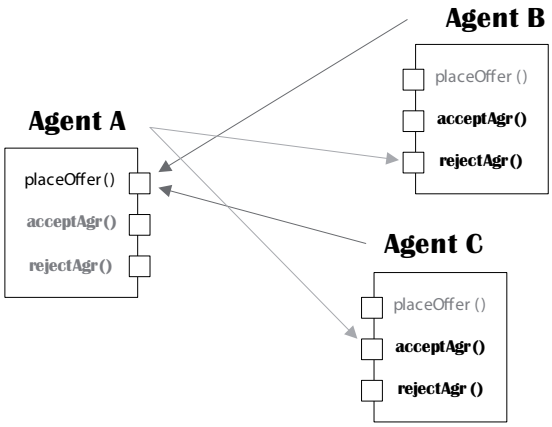


Figure 4. Auction Process

4 Negotiation Protocol

After supplying all negotiation participants with the protocol description, the actual negotiation can start. We propose a generic negotiation protocol able to cope with the different negotiation types that can be expressed with the means of our protocol description documents.

In general, every negotiation is specified as a bidding process. Each party involved in a negotiation offers an agreement to the other party concerning the issues subject to the negotiation that is currently acceptable for them. Then the other party assesses the offered agreement and generates a counter-offer, accepts the offer or rejects it and terminates the negotiation. This way the two parties (service consumer and provider) involved move from a conflict situation concerning some (logical) resource(s) to a consensus represented by the resulting agreement. In order to support such processes our generic negotiation protocol has to provide the agents with means to post offers and to promote the decision made about a concrete offer.

The two roles present within the actual negotiation protocol are the *Negotiation Participant* and the *Information Service*. Given its role during a negotiation the Negotiation Participant interface offers methods for placing offers as well as methods for accepting and rejecting offered agreements.

- *placeOffer(agentEPR, WS-Agreement-document)*
- *acceptAgreement(negotiationID, agreementID)*
- *rejectAgreement(negotiationID)*

The Information Service role provides access to information on the current negotiation status or past offers. Hence the corresponding interface provides the following methods:

- *getStatus(negotiationID)*
- *getPastOffers(negotiationID)*/ *getPastOffers(negotiationID, agentID)*

The *getStatus*-method is used by all negotiation participants to access the current negotiation status. This allows, for example, to assess which offer is currently winning the negotiation

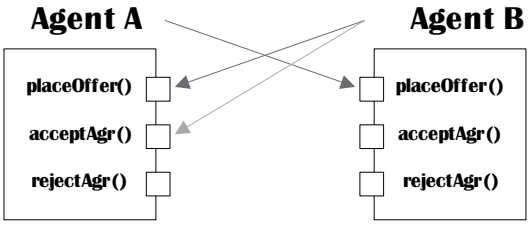


Figure 5. Bargaining Process

and if necessary to adopt the own offer. The remaining methods let participating agents access past offers of a negotiation. This information can be used for internal decision making of the negotiating agents. Such a request may be restricted to offers posted by a specific agent denoted by its ID as an additional parameter.

Figures 4 and 5 schematically show how the interfaces are used during an auction and a bargaining scenario.

It should be noted that currently only a polling mechanism is available for accessing negotiation related information. Other concepts of information distribution, such as publish/subscribe and notification functionality for a more flexible information processing mechanism, are explored at the moment.

Using these roles and interfaces a rather generic negotiation protocol is defined, capable of conducting any negotiation protocol describable with the attributes identified before. Agents allowed to post bids (stated in the protocol description) do so via the *placeOffer()*-method offered by the respective negotiation partner (the auctioneer in auctions or both agents in bargaining protocols offer this method). When terminating a negotiation each participant is notified using the *accept/rejectAgreement()*-methods. Additionally the negotiators can query negotiation related data at the Information Service.

5 Conclusion

This paper proposes a negotiation framework for WS-Agreement, enabling the integration of a variety of negotiation protocols suitable for different application domains based on an exchange protocol determining the actual negotiation protocol used. Negotiation protocols can be specified in a description language and made available to parties interested in negotiations. Parties interested in negotiating an agreement first run the negotiation exchange protocol to establish which negotiation protocol is used. Subsequently, the protocol is executed to determine the resulting, negotiated WS-Agreement document. Finally, after winner determination, acceptance and rejection is performed again according to the standard WS-Agreement protocol. With these two protocols fully automated WS-Agreement negotiations according to a variety of different negotiation protocols can be conducted in Web Service environments.

Future work focuses on testing a variety of negotiation protocols, e.g., service level agreements in the context of grid environments, such as [12], and thus verifying the expressiveness of the negotiation description language and the capabilities of the exchange protocol.

References

[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification draft, version 09/2005. 2005.

[2] C. Bartolini, C. Preist, and N. R. Jennings. A software framework for automated negotiation. *Software Engineering for Multi-Agent Systems III (eds. R. Choren, A. Garcia, C. Lucena, and A. Ramonovsky)*, pages 213–235, 2005.

[3] M. Bichler and J. R. Kalagnanam. Software frameworks for advanced procurement auction markets. *Communications of the ACM (CACM)*, 49(12), 2006.

[4] M. B. Chhetri, J. Lin, S. Goh, J. Y. Zhang, R. Kowalczyk, and J. Yan. A coordinated architecture for the agent-based service level agreement negotiation ofweb service composition. In *Proc. of the Australian Software Engineering Conference (ASWEC'06)*, pages 90–99, Washington, DC, USA, 2006. IEEE Computer Society.

[5] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In *JSSPP 2002*, Vol. 2537 of *LNCS*, pages 153–183. Springer, 2002.

[6] F. D'Andria, J. Martrat, P. Laria, G. Ritrovato, and S. Wesner. An enhanced strategy for sla management in the business context of new mobile dynamic vo. In *Proceedings of the 4th eChallenges Conference*, 2006.

[7] S. Hudert. A proposal for a web services agreement protocol framework. Bamberger Beiträge zur Wirtschaftsinformatik 70, Bamberg University, February 2007. ISSN 0937-3349.

[8] A. R. Lomuscio, M. Wooldridge, and N. R. Jennings. A classification scheme for negotiation in electronic commerce. *Int. J. of Group Decision and Negotiation*, 12(1):31–56, 2003.

[9] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. Web service level agreement (wsla): Language specification version (1.0), January 2003.

[10] E. D. Nitto, M. D. Penta, A. Gambi, G. Ripa, and M. L. Villani. Negotiation of service level agreements: An architecture and a search-based approach. In *Intern. Conf. on Service Oriented Computing (ICSOC) 2007*, pages 295–306, 2007.

[11] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, K. Krishnakumar, and A. Meisels. A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing. *Advances in Grid Computing - EGC 2005*, 3470/2005:651–660, 2005.

[12] SORMA. EU Information Society Technologies project SORMA - Self-Organizing ICT Resource Management. 2007.

[13] M. Stroebel and C. Weinhardt. The montreal taxonomy for electronic negotiations. *Journal of Group Decision and Negotiation*, 12:143–164, 2003.

[14] P. R. Wurman, M. P. Wellman, and W. E. Walsh. The michigan internet auctionbot: A configurable auction server for human and software agents. *Prof. of the Second Intern. Conf. on Autonomous Agents*, May 1998.

[15] P. R. Wurman, M. P. Wellman, and W. E. Walsh. A parametrization of the auction design space. *Games and Economic Behavior*, 35(1-2):304–338, 2001.

Taxonomy on Consistency Requirements in the Business Process Integration Context

Andreas Schönberger and Guido Wirtz
Distributed and Mobile Systems Group, University of Bamberg
Feldkirchenstr. 21, 96052 Bamberg, Germany
E-mail: {andreas.schoenberger | guido.wirtz}@uni-bamberg.de

Abstract

Consistency is a general goal in software development. The development of B2B integration (B2Bi) software has extended consistency requirements (CR) because B2Bi projects require personnel from different organizations to agree about the what and how of integrations and these projects require distributed computing as central IT infrastructure frequently is not available. The rise of numerous approaches targeting at consistency in the B2B area are evidence of these enhanced CRs. This paper refines the Open-edi business transaction model into a B2Bi schema by analyzing abstraction levels, development phases and distribution aspects of B2Bis. A taxonomy of CRs is then derived accordingly. Thus, this paper underlines the extended importance of consistency in B2Bis, furnishes a criterion for choosing B2Bi methods, helps classifying consistency support in B2B approaches and gives a starting point for finding consistency checking approaches.

Keywords: consistency, compatibility, business process integration, business collaboration, SOA

1. Introduction

Consistency is a predominant requirement in software development. This holds true for the early design phases of software development where methods like model checking (cf. [1]) are applied, for the transition between development phases (cf. [2]) and particularly for ensuring operational consistency using transactions in databases or transaction monitors.

The B2Bi domain even has extended CRs. First, this becomes clear by looking at the special organizational and technical conditions in the B2Bi domain. Organizationally speaking, people from different organizations with possibly different background and vocabulary have to agree about what goals to achieve in an integration and have to develop the necessary interaction protocols therefore. Technically speaking, central technical infrastructure is frequently not available or prohibited by business politics so that truly distributed computing is needed. Second, in order to provide consistent change of the common business state of integration partners, standards for realizing transactions in the Web Service domain are being developed, e.g.,Web Services Atomic Transaction (WS-AtomicTransaction, [3]) and Web Services Business Activity (WS-BusinessActivity, [4]) which are both constituent parts of the OASIS Open Web Services Transaction specification[1]. Taking into account that Web Services are an important technique for implementing B2Bis the development of these standards also bears witness to the special consistency needs of B2Bi.

Therefore, in this paper, we develop a B2Bi schema by analyzing the abstraction levels of B2Bi projects, its development phases and the purpose of relevant B2Bi standards. We then identify CRs between the components of this schema and thus derive a taxonomy of CRs. Note, that this paper does not define a new notion of consistency. For the purpose of identifying CRs the following, rather general, standard definition is used instead so as to capture a large amount of CRs: "**consistency.** *The degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component.*" [5].

Finally, after having discussed related work, we conclude the paper and point out directions for future work.

2. A B2B integration schema

The analysis of CRs in B2Bis needs a conceptualization of the domain in order to define between which concepts consistency is required. The development of our B2Bi schema therefore considers three important aspects which lay behind the need for consistency.

- *Abstraction levels.* B2Bis can be viewed on several abstraction levels where these abstraction levels should be consistent with each other.
- *Development phases.* B2Bis are developed according to some Software Engineering Process (SWE) of choice which consists of several phases. The artifacts produced during these phases should be consistent with each other.

[1] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-tx

- *Organizational distribution.* Multiple parties are interacting in B2Bis. These parties usually have their own IT infrastructure and IT policies. So the techniques and tools used for implementing B2Bis have to consider interoperability between the partners and transactions crossing enterprise boundaries.

Clearly, these three aspects overlap, but this distinction is useful for identifying gaps in the derivation of the schema according to only one aspect.

The derivation of our B2Bi schema starts out by considering the Open-edi reference model [6] which looks at business transactions on two abstraction levels. The so-called *Business Operational View* (BOV) covers the business aspects of business transactions while the so-called *Functional Service View* (FSV) covers the information technology aspects of business transactions [6]. This model, used as a B2Bi schema for identifying CRs, already furnishes a basic CR of the FSV being consistent with the BOV but this obviously is far too general for a CRs taxonomy of B2Bis.

The Open-edi reference model of business transactions is further refined by [7] for the purpose of classifying current B2B methodologies and technologies. This refinement splits up the BOV into *business models* {A} and *business process models* {B} while the FSV is split up into *deployment artifacts* {C} and *software environments* {D} (see figure 1 taken from [7]).

According to [7], business models describe the exchange of

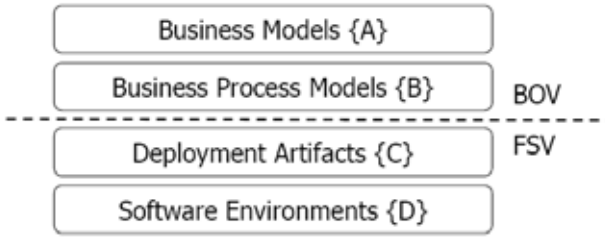


Figure 1. Classification schema based on refinements of the Open-edi reference model [7]

values between partners on an abstract level whereas business process models detail the relationships between the partners by specifying the flow of information and type of interaction. We agree with the authors in this point and also keep these two levels for our B2Bi schema. Further, [7] paraphrase deployment artifacts as machine-processable descriptions of business transactions and software environments as concrete implementations of information systems. While this distinction may be sufficient for surveying B2B methodologies and technologies we refine the schema of [7] in two points.

First we claim that, for a CRs taxonomy, the deployment artifacts level should be separated in a so-called *choreography* and a so-called *orchestration* layer. This finer distinction is necessary for respecting the distribution of the collaborating partners adequately, so that the overall message exchanges can be specified on the choreography level while the message exchanges of a single partner can be specified on the orchestration level. Apart from this argument, the development of choreography standards like WS-CDL [8] and ebxml BP (formerly known as BPSS, [9]) as well as orchestration standards like WS-BPEL [10] evidence the need for this distinction. Moreover, the orchestration level should be split up into so-called *public processes* and so-called *private processes*. This separation pays tribute to the obligation of an integration partner to obey a particular externally observable message sequence (public process) and, at the same time, integrate this public process with its (preexisting) backend systems (private process). One could argue, that if the participation of multiple integration partners leads to the dichotomy of choreography and orchestration, then the business process models would also have to be divided into *global* and *local* business process models. We claim that, from a B2Bi point of view, the core task of business process models is providing a means of communication for agreement of how to achieve business goals and we also developed a modeling approach for this task [11, 12, 13]. Nonetheless, B2Bi process models may be enhanced by local process models when doing local optimizations which then would introduce new CRs not discussed here.

Second, we refine the concept of software environments in [7] as we do not simply consider these to be implementations of information systems but to be the source for tracing consistency between actual process executions and process specifications. Hence we rename the software environments level as *runtime systems* furnishing the raw data for checking conformance of process executions with process specifications.

Although the B2Bi schema developed so far already lays the foundation for finding very important CRs, we enhance it by the following findings.

Looking at software development processes in general and in particular at the system/software requirements engineering phase, the lack of the *real world* in the schema is apparent.

Keeping the distribution of integration partners in mind, it is also clear that not only the message exchanges of the integration partners matter but also a way for synchronizing their local views on the global state of the B2Bi is essential. Thus, a way for implementing distributed transactions has to be found. A proof for this necessity is the development of standards like WS-AtomicTransaction [3] and WS-BusinessActivity [4]. Apart from distributed transactions, the integration partners have to agree about and provide for Quality-of-Service (QoS) aspects of their collaboration. Regarding the provision of QoS aspects, the integration partners either have the option to use the same integration frameworks or to use interoperability standards for ensuring QoS aspects like WS-Reliability [14] or WS-Security [15]. Thus, the application and the interoperability of such standards is another source for the identification of CRs.

The results of this discussion are summarized by figure 2 which does not only visualize the B2Bi schema developed but also identifies relevant CRs which are detailed in the next section.

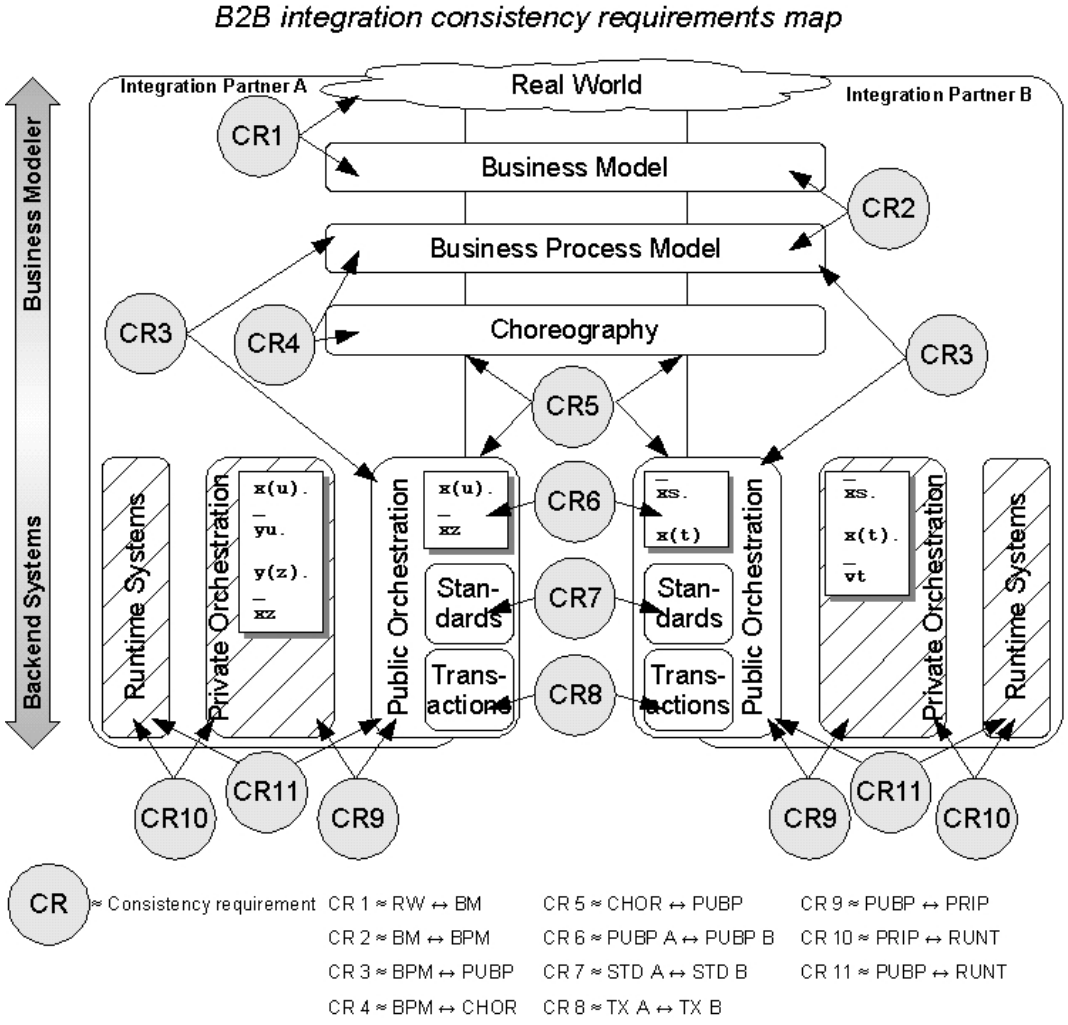


Figure 2. Consistency requirements

3. Consistency requirements

The identification of CRs using the given B2Bi schema is achieved by analyzing the relations between the components of the schema where each component contains a set of models for representing the B2Bi on a particular abstraction level. CRs between two schema components are actually CRs between the models of different components and usually emerge if these models are produced in subsequent phases of a particular software engineering process (SWE) used for implementing the B2Bi. Clearly using this way of identifying CRs would be heavily dependent on particular SWEs but it helps at least eliminating obscure CRs that would emerge from no reasonable SWE, e.g., no reasonable SWE would propose the definition of WS-BPEL [10] processes as the first step in implementing a B2Bi. Further, following the phases of a SWE, the models become more and more detailed. These additional informations cannot be completely derived from more abstract models because the more concrete models wouldn't present any further information otherwise. Theoretically this would lead to a further CR between any schema component and the real world but this theoretical requirement is neglected for practicability reasons. Finally, there are also CRs between the models of a single level, intra-model CRs (e.g. syntactic conformance to modeling languages used) and *evolution consistency* [26] requirements but these requirements are neglected here because they are a different area of concern. Applying this approach for identifying CRs leads to the set of CRs depicted in figure 2. The naming of the CRs is derived from the schema components that give rise to those CRs. This naming convention pays tribute to the fact that we are talking about consistency *requirements* and not about specialized definitions of consistency like *process inheritance* or *compatibility* [2] although these notions clearly may be related to our CRs. For each of these CRs we were able to find at least one approach that supports it which constitutes empirical evidence for the existence of the respective CR. Moreover, the solutions of the approaches under study could all be mapped to our CRs which may be a hint that there are not too many CRs missing in our taxonomy. As a detailed discussion of the approaches found must be omitted due to space limitations we provide a classification in table 1 that relates the approaches (Appr.) to the CRs of our taxonomy. Note that CR 10 and CR 11 are merged to one column and that we have included

Appr.	Consistency requirement										A/C
	RW ↔ BM	BM ↔ BPM	BPM ↔ PUBP	BPM ↔ CHOR	CHOR ↔ PUBP	PUBP A ↔ PUBP B	STD A ↔ STD B	TX A ↔ TX B	PUBP ↔ PRIP	SPEC ↔ RUNT	
[16]	+	-	-	-	-	-	-	-	-	-	A
[17]	-	+	-	-	-	-	-	-	-	-	C
[11, 13]	-	+	+	-	-	-	-	-	-	-	A/C
[18]	-	-	+	-	+	+	+	-	+	+	A
[19]	-	-	-	+	-	-	-	-	-	-	C
[20]	-	-	-	+	+	-	-	-	-	-	A/C
[21]	-	-	-	-	+	-	-	-	-	-	A
[22]	-	-	-	-	-	+	-	-	+	-	A
[23]	-	-	-	-	-	-	+	-	-	-	A
[3, 4]	-	-	-	-	-	-	-	+	-	-	C
[24]	-	-	-	-	-	-	-	-	+	-	A
[25]	-	-	-	-	-	-	-	-	-	+	A

Table 1. Survey of approaches targeting at consistency and CRs supported

an extra column that describes whether a particular approach supports consistency by means of analyzing models (A) or by constructing models (C) or both (A/C). In the following, we discuss the CRs of our taxonomy in more detail:

CR 1: Real World (RW) ↔ Business Model (BM). This requirement is sometimes overlooked because the BM may not be part of the regular SWE artifacts produced during a B2Bi or the BM is quite nontechnical. If modeled, the BM cannot be checked for consistency with the RW automatically. However the BM itself is at least amenable to automatic analysis and thus inconsistencies within the BM may reveal inconsistencies between the RW and the BM.

CR 2: Business Model (BM) ↔ Business Process Model (BPM). The task of the BPM is to specify how to achieve the business goals defined in the BM by defining the types of information to be exchanged, the flow of information and organizational aspects. CR 2 demands, that these specifications ensure the business goals or at least don't contradict them. One could argue that there should also be a CR between RW and BPM in case there's no BM. We do not reject this argument but leave out this CR for practical reasons.

CR 3: Business Process Model (BPM) ↔ Public Processes (PUBP). While the BPM is a *common* model the integration partners have to agree upon, the PUBP is the definition of the communication tasks of each partner. If BPM is transformed into PUBP directly, CR 3 demands that the PUBP strictly conform to the BPM. This conformance is achievable with respect to control flow but it is also necessary to specify Quality-of-Service (QoS) attributes of the PUBP which frequently lack in BPM, e.g., *Reliable Communication* or *Security*. These QoS attributes can be more easily checked for conformance if BPMs are not directly transformed into PUBP but first in CHOR and afterwards in PUBP.

CR 4: Business Process Model (BPM) ↔ Choreography (CHOR). The differences between BPMs and CHOR standards are sometimes fluent but it helps to think of BPM as a model that serves as communication means between business analysts while CHOR is a detailed technical communication specification intended to be processed by machines. The main claim of CR 4 is the conformance of information types and control flow in CHOR to the same aspects in BPM while the aforementioned QoS aspects frequently have to be introduced in CHOR.

CR 5: Choreography (CHOR) ↔ Public Processes (PUBP). It is feasible to generate PUBP from a CHOR specification to a large extent. But taking into account that the results of this transformation are not unique, the demand of CR 5 for conformance of information types, control flow and QoS aspects should not be neglected, in particular for bottom-up approaches.

CR 6: Public Process of partner A (PUBP A) ↔ Public Process of partner B (PUBP B). CR 6 defines the first of three CRs between arbitrary integration partners who are simply referred to as *partner A* and *partner B* for the sake of practicability. CR 6 particularly refers to the compatibility (as a form of consistency) between the observable communication of the integration partners. Note, that compatibility checks for particular properties by analyzing the PUBP may be replaced by checking the properties for the CHOR and then proving that the transformation of CHOR into PUBP is preserving these properties.

CR 7: Standards used by partner A (STD A) ↔ Standards used by partner B (STD B). CR 7 takes into account that the partners of B2Bis frequently are independent from each other and thus may have heterogeneous IT systems. In order to provide for the correct implementation of QoS attributes like *Reliable Messaging* or *Security* the integration partners then either have to use the same integration frameworks with the same configurations or they have to agree on the application of QoS interoperability standards like [14] or WS-Security [15]. The former approach leads to tight coupling between IT systems and is more and more unacceptable nowadays.

CR 8: Transactional data concerned by partner A (TX A) ↔ Transactional data concerned by partner B (TX B). CR 8

alludes to the common state of a business collaboration that has to be synchronized among the integration partners, e.g.,whether an order has been accepted or not or if a bill has been delivered or not. Standards like [3] and [4] have been developed to meet this requirement. In order to decide if an information item is to be synchronized using these standards it has to be decided if it belongs to the common business state.
CR 9: Public Processes (PUBP) ↔ Private Processes (PRIP). Frequently PRIP are just a refinement of PUBP that couple the observable communication of an integration partner to its backend systems. CR 9 claims that these refinements may not change the observable communication as otherwise the interaction protocol of the collaborating partners would be broken. The analysis of conformance of PRIP to PUBP is especially useful in bottom-up approaches where preexisting private processes may be wrapped to conform to predefined public processes.
CR 10: Private Processes (PRIP) ↔ Runtime Systems (RUNT). CR 10 claims that a process instance of an integration partner in execution must conform to its specification. This CR is usually met by installing monitoring systems for RUNT that furnish sufficient information. Note, that if an executed process conforms to its private process specification and CR 9 is sufficiently accounted for, then CR 11 is met as well.
CR 11: Public Processes (PUBP) ↔ Runtime Systems (RUNT). Although CR 11 may be substituted by CR 9 and CR 10, CR 11 is stipulated because it may be far easier to check because of the possibly reduced state space of public processes compared to private processes. From the B2Bi point of view both options ensure that a process in execution does not break the interaction protocol of the integration partners so just addressing CR 11 and ignoring CR 10 may be admissible as well.
The core contribution of this taxonomy is the identification of high level CRs that emerge during B2Bi projects. During such a project, a particular CR has to be further investigated with respect to an adequate definition of consistency, evaluation of suitable modeling methods and analysis tools as well as organizational implications. Finally, we claim that the choice of B2Bi frameworks should consider the support of the CRs identified.

4. Related Work

Clearly, consistency always played an important role in software development [2, 27, 28, 29] but we put our focus particularly on the B2Bi domain. In that domain, there is a lot of literature that discusses methods for checking some kind of consistency which frequently apply concepts like *process inheritance* or *process compatibility*. But work about CRs in the B2Bi domain is very scarce. Greenfield et al. [30] discuss *Consistency for Web Services Applications*. Their work is different from ours in so far as they discuss in detail for Web Services what we identified as CR 6 (PUBP A ↔ PUBP B) and CR 8 (TX A ↔ TX B). Decker et al. [31] describe compatibility and consistency notions in the B2Bi domain. They define consistency between public and private processes (CR 9) with respect to compatibility between public processes of interacting parties (CR 6, CR 5). In so far they also focus on parts of our taxonomy. To our knowledge we are the first to derive a detailed CRs taxonomy for the B2Bi domain regarding several abstraction levels.
Apart from that, there are several papers that discuss the comparison between various business process reference models or business process modeling methodologies like [32] or [33]. Our taxonomy is not suitable for performing such comparisons but we claim that consistency should be an important criterion in these comparisons and that our taxonomy is useful for classifying and comparing business process reference models and business process modeling methodologies with respect to support for CRs.
Finally, there is extensive work on *(in)consistency management* [27, 29] that describes how and when to enforce consistency and how to react to inconsistencies. Spanoudakis and Zisman [27] propose a process that consists of *detecting overlaps in models*, *detecting, diagnosing, handling and tracking inconsistencies*, as well as *specifiying and applying a management policy for inconsistencies*. Although consistency management is different from the work presented here the CRs identified can help in deciding where to apply processes like the one described by [27].

5. Conclusions and Future Work

In this paper we systematically derived a taxonomy of CRs using a B2B schema. The various areas that call for consistency checking methods underpin the importance of consistency in the B2Bi domain and especially necessitate the consideration of consistency in comparison frameworks for B2Bi methodologies. Our taxonomy is also useful for classifying approaches targeting at consistency checking and the survey we undertook not only proves empirical evidence for the existence of our CRs but also gives a starting point for finding relevant consistency checking methods.
In the future, each CR should be analyzed in more detail in order to develop differentiated criteria for evaluating support for a particular CR by a particular consistency checking method. Looking at the diversity of CRs identified, the need for integrated consistency support throughout the whole life cycle of B2Bis, i.e. the application of methods like (in)consistency management [27, 29], is striking. In particular, integrating consistency management practices into SWEs targeting at B2Bi is an interesting area of research. In this respect, the seamless application of existing consistency checking methods throughout several abstraction levels as well as enhancing the usability of rather scientific approaches is also an interesting area of research. Finally, special attention from the point of view of consistency is to be payed to the question whether BPMs should be directly mapped to public processes or indirectly via choreography specifications.

References

[1] E. M. Clarke Jr., O. Grumberg, and D. A. Peled, *Model checking*. Cambridge, MA, USA: MIT Press, 1999.

[2] C. Canal, E. Pimentel, and J. M. Troya, "Compatibility and inheritance in software architectures," *Sci. Comput. Program.*, vol. 41, no. 2, pp. 105–138, 2001.

[3] OASIS Open, "Web Services Atomic Transaction (WS-AtomicTransaction) version 1.1," July 2007.

[4] ——, "Web Services Business Activity (WS-BusinessActivity) version 1.1," July 2007.

[5] IEEE, *IEEE Standard Glossary of Software Engineering Terminology*, 1990.

[6] ISO/IEC, *Information technology - Open-edi reference model*, 2nd ed., ISO/IEC, May 2004.

[7] Dorn et al., "A survey of B2B methodologies and technologies: From business models towards deployment artifacts," *HICSS*, vol. 00, p. 143a, 2007.

[8] W3C, *Web Services Choreography Description Language*, 1st ed., W3C, November 2005.

[9] OASIS Open, *ebXML Business Process Specification Schema*, 2nd ed., OASIS Open, December 2006.

[10] ——, *Web Services Business Process Execution Language*, 2nd ed., April 2007.

[11] A. Schönberger and G. Wirtz, "Using Webservice Choreography and Orchestration Perspectives to Model and Evaluate B2B Interactions," in *SERP 2006*, June 26-29 2006.

[12] ——, "Realising RosettaNet PIP Compositions as Web Service Orchestrations - A Case Study," in *EEE 2006*, June 26-29 2006.

[13] A. Schönberger, "Modelling and Validating Business Collaborations: A Case Study on RosettaNet," University of Bamberg, Contributions to Applied and Business Informatics of University of Bamberg 65, Mar. 2006.

[14] OASIS Open, "WS-Reliability v1.1," November 2004.

[15] ——, "Web Services Security v1.1," February 2006.

[16] OMG, *Semantics of Business Vocabulary and Business Rules (SBVR), v1.0*, OMG, January 2008.

[17] Koliadis et al., "Combining i* and BPMN for business process model lifecycle management," in *Business Process Management Workshops*, 2006, pp. 416–427.

[18] L. Baresi and S. Guinea, "Towards dynamic monitoring of WS-BPEL processes," in *ICSOC*, ser. LNCS, vol. 3826. Springer, 2005, pp. 269–282.

[19] M. Ilger and M. Zapletal, "An implementation to transform business collaboration models to executable process specifications," in *Service Oriented Electronic Commerce*, 2006, pp. 9–23.

[20] Zhao et al., "Towards the formal model and verification of web service choreography description language," in *WS-FM*, ser. LNCS, vol. 4184, 2006, pp. 273–287.

[21] W. Yeung, "Mapping WS-CDL and BPEL into CSP for behavioural specification and verification of web services," *ecows*, vol. 0, pp. 297–305, 2006.

[22] Weidlich et al., "Efficient analysis of BPEL 2.0 processes using p-calculus," *Asia-Pacific Service Computing Conference*, pp. 266–274, 11-14 Dec. 2007.

[23] Nezhad et al., "Web services interoperability specifications," *Computer*, vol. 39, no. 5, pp. 24–32, May 2006.

[24] A. Martens, "Consistency between executable and abstract processes," *e-Technology, e-Commerce and e-Service, 2005*, pp. 60–67, April 2005.

[25] W. van der Aalst, "Business alignment: using process mining as a tool for delta analysis and conformance testing," *Requirements Engineering Journal*, vol. 10, pp. 198–211, August 2005.

[26] Engels et al., "Towards consistency-preserving model evolution," in *IWPSE '02*. ACM, 2002, pp. 129–132.

[27] G. Spanoudakis and A. Zisman, *Handbook of Software Engineering and Knowledge Engineering*. World scientific, 2001, ch. Inconsistency management in software engineering, pp. 329–380.

[28] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp. 70–93, 2000.

[29] R. Van Der Straeten, "Inconsistency management in model-driven engineering," Ph.D. dissertation, Vrije Universiteit Brussel, Belgium, September 2005.

[30] Greenfield et al., "Consistency for web services applications," pp. 1199–1203, 2005.

[31] G. Decker and M. Weske, "Behavioral consistency for B2B process integration," in *CAiSE*, ser. LNCS, vol. 4495. Springer, 2007, pp. 81–95.

[32] Fettke et al., "Business process reference models: Survey and classification," in *Business Process Management Workshops*, vol. 3812, 2005, pp. 469–483.

[33] Kaschek et al., *Technologies for Business Information Systems*. Springer, 2007, ch. Characterization and Tool Supported Selection of Business Process Modeling Methodologies, pp. 25–37.

Developing Enterprise Applications with Support to Dynamic Unanticipated Evolution

Hyggo O. de Almeida, Marcos F. Pereira, Márcio de M. Ribeiro, Angelo Perkusich, Emerson Loureiro and Evandro Costa

Abstract— **This paper presents a component based framework for developing enterprise applications with support to dynamic unanticipated evolution. The framework is based on the COMPOR Component Model Specification, which provides mechanisms to manage unpredicted evolution even at runtime. We describe the framework design that is based on design patterns and aspect-oriented concepts. Finally, we present a example application of the framework in the context of electronic commerce.**

Index Terms— **Unanticipated Software Evolution, Enterprise Applications, Component-Based Development**

I. Introduction

Enterprise information systems are applications for handling company-wide information and delivering services to a wide range of users. Such systems must be: secure, to protect users and the enterprise; scalable, to ensure that users simultaneously take advantage of various services; and reliable, to ensure the consistency of the transactions processing.

Besides these features, enterprise applications change frequently. Considering the complexity of these applications, requirement changes cause a great impact on the system architecture, design and code. This impact is even more relevant when such changes are not predicted at design time. Unanticipated changes have been pointed out as the main reason of problems related to software evolution activities [1]. In the case of enterprise applications that cannot be interrupted for financial or safety reasons, it becomes even more difficult to manage unanticipated evolution at runtime.

J2EE [2] and .NET [3] are well known platforms for developing and deploying enterprise applications. Developers using such platforms save time by not looking at a diverse range of products and services, since they are already provided by those platforms. Such services include security, persistence, distribution, load balancing, and transaction management, among others. Nevertheless, J2EE and .NET do not support adequately dynamic unanticipated software evolution. This occurs due to the high coupling among components, which makes difficult to implement unpredicted changes on the fly.

To deal with this problem, in [4] is proposed a component model to develop software supporting dynamic unanticipated evolution named COMPOR Component Model Specification (CMS). Such a model allows changing any part of the software, by removing and/or adding components, even at runtime. It is also proposed a Java implementation of CMS called Java Component Framework (JCF), which is used to develop Java applications supporting dynamic unanticipated evolution.

However, the JCF framework do not provide support for developing enterprise applications. When developing software with JCF, developers have to implement all features related to enterprise applications, such as security, distribution and transactions management from scratch. In this paper we introduce an extension of JCF framework for developing enterprise applications with support to dynamic unanticipated evolution. More specifically, we describe how to extend JCF design to implement distribution, security and transaction management by using design patterns and aspect-oriented programming.

The remainder of this paper is organized as follows. In Section II, we present the extension for enterprise applications. Section III describes a example application of the framework. Section IV discusses some related works. Finally, in Section V, we present the final remarks.

II. Support for Enterprise Application

In this section we present the extension of JCF to develop enterprise applications. More specifically, we describe how to extend JCF to provide support for security, transaction management and distribution features.

A. Security

According to CMS, an alias is used to uniquely identify services and events with the same name for different components. However, such a strategy introduces a security problem into the model. For example, it is possible to interpose a provider *X* between another provider *Y* and its clients in order to intercept the client requests towards *Y*. This may represent an intrusive way to make something undesirable in the system, since the interposed provider *X* may be seen as an intruder.

As this security issue is not tackled by the component model, the JCF must provide means for dealing with security policies for the interaction and deployment models. Such policies must then be satisfied when some service is requested or an event is announced as well as a component is inserted into or removed from a container. This security infrastructure, shown in Figure 1, was developed using aspect oriented

The authors are with the Embedded System and Pervasive Computing Laboratory, Department of Electrical Engineering, Federal University of Campina Grande, C.P. 10105 - 58109-970 - Campina Grande - PB - Brazil, emails: hyggo@dsc.ufcg.edu.br, marcos@embedded.ufcg.edu.br, mmr3@cin.ufpe.br, perkusic@dee.ufcg.edu.br, evandro@ic.ufal.br

programming, with AspectJ [5]. Aspects have allowed to hide the complexity of the security mechanism from the developer as well as to simplify the development of systems without security requirements. The security mechanism illustrated in Figure 1 is explained as follows.

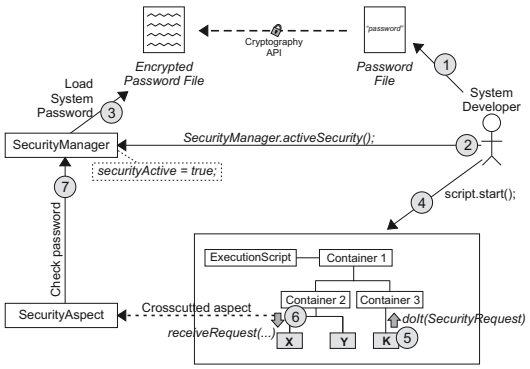


Fig. 1. Aspect oriented security architecture.

1) The application developer creates a ".security" file containing the password for accessing the system as well as the service access policies. Then, uses the Java cryptography API to encrypt the file.
2) When developing the application, the security mechanism should be activated calling the `activeSecurity()` method of the `SecurityManager` singleton class. This operation defines that all service invocations, event announcements and component additions must be verified.
3) The `SecurityManager` retrieves the password and the policy information and stores them in memory.
4) After starting the root container, all of its components are also started and the application runs by means of a sequence of service invocations and event announcements.
5) A component invokes a service. With the security activated, the service requester component must forward a `SecurityServiceRequest` instance as parameter, containing the system password.
6) The component receives the request via the `receive Request` method, then the `SecurityAspect` aspect intercepts the method invocation and asks the `Security Manager` to verify the request password.
7) `SecurityManager` verifies the request password and allows the service execution. Otherwise, a `ComporSecurity Exception` is thrown.

B. Transaction

Figure 2 illustrates the component *K* requiring the execution of two services. The first (*withdraw*) is implemented by the component *X*, whereas the second (*deposit*) is implemented by the component *Y*. Because it represents a money transferring, such operation must be atomic (or indivisible), which means that the money either moves between the two accounts or it stays in the first account.

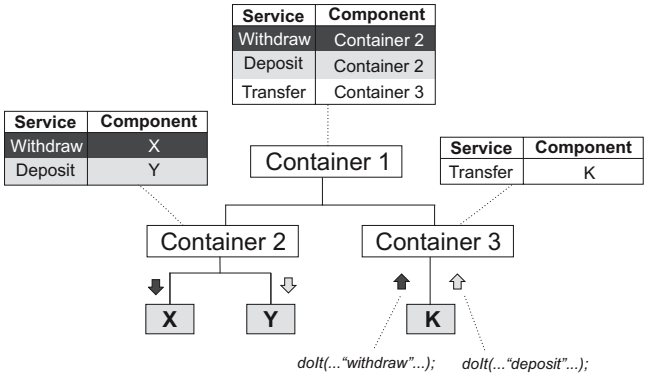


Fig. 2. Necessity of atomic operations.

In order to handle with atomic situations, a transaction mechanism is available. Such mechanism implements the two-phase commit protocol, a popular protocol used to guarantee consensus between the participating members of a transaction [6].

In the same way of the security mechanism, the Aspect-Oriented approach was used, allowing us to separate the transaction concern as well as to develop systems without it by simply removing the aspect responsible for implementing the mechanism. Therefore, the simplicity of the CMS model was maintained, since it does not depend on the transaction mechanism.

The two-phase commit protocol defines a coordinator that is responsible for governing the outcome of the transaction. In the first phase of the protocol, the participants (in our case, components) must invoke their init service. According to the all participants answers, in the second phase the coordinator decides whether it will commit or rollback the transaction by sending a message with its decision to all participants.

According to the CMS model, when clients invoke services, they must use instances of the *ServiceRequest* class. However, if clients have to execute transactional services, they must use instances of the *TransactionServiceRequest* class instead. Notice that a such class extends *ServiceRequest*.

Each CMS component must extend the *FunctionalComponent* class, which has an important method named *receiveRequest* [4]. Since *receiveRequest* is called by the framework before the execution of services, the aspect responsible for the transaction mechanism verifies the instance of the service request. If the request is an instance of the *ServiceRequest* class, the service is executed normally. Otherwise, a transaction is started.

Notice that the verification about which service (init, commit, or rollback) will be executed is weaved by the aspect in the *receiveRequest* method. Hence, the implementation of the protocol is guaranteed by this verification. In addition, the consistence of information through atomic operations is guaranteed as well.

Aiming at completing the ACID properties, the mechanism also provide isolation of transactions through synchronization

of threads. Besides, in order to guarantee the consistence of data in case of hardware crashes, each transaction is logged. When the system comes back, the aspect crosscuts its initialization and recovers the transactions automatically through the log file reading.

C. Distribution

Distribution is a desirable feature for enterprise applications, since it might provide performance increasing, economies of scale, reliability (if carefully designed), and resource sharing (through the use of a computer network).

When considering distributed software, each module of the software might reside in different computers in the network. The communication among those modules is based on sending messages to each other. In the component based development context, these modules consist of components of software.

Similarly to the J2EE and CORBA, JCF containers play a fundamental role in the distribution implementation as well. In this context, each JCF container extension is responsible for sending requests and event announcements to their distributed components children. Figure 3 illustrates the distribution mechanism, which relies on the Decorator [7] and Proxy [7] design patterns. Notice that it is an extension of the CMS model. This way, the simplicity of the model remains, since it does not depend on the referred mechanism.

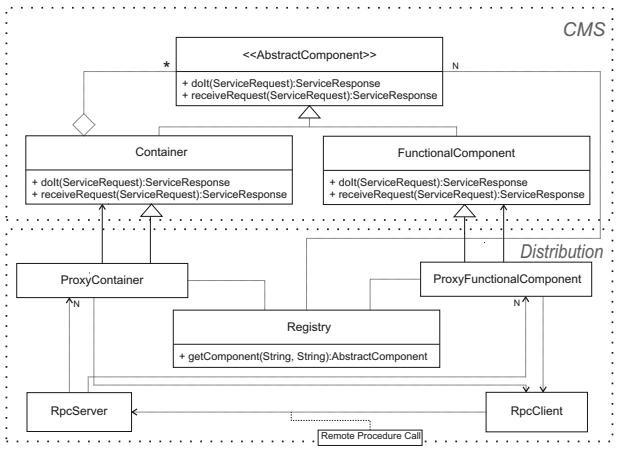


Fig. 3. Distribution architecture.

The architecture of the distribution mechanism is explained as follows. Containers have children which consist of representative entities (relying in Proxy [7] implementation). Notice that such entities point out to the remote functional component and the parent of the remote components is also a representative entity. Nevertheless, it points out to the remote container instead.

In order to get started with a distributed application, as illustrated in Figure 4, the application developer must deploy the desired part of the hierarchy into each participant host. For each host, such a developer must execute the following steps to configure the distribution:

1) In the host *192.168.10.6*, he must add an instance of `ProxyFunctionalComponent` retrieved from the host *192.168.10.1* (by a remote procedure call), such instance is a proxy to the real component which resides in the host *192.168.10.1*. Notice that the real component is child of the container localized in the host *192.168.10.6*.
2) In the host *192.168.10.6* the real component is added as a child of the `ProxyCont1`, which is a `ProxyContainer` instance retrieved from the host *192.168.10.6* (also a remote procedure call).

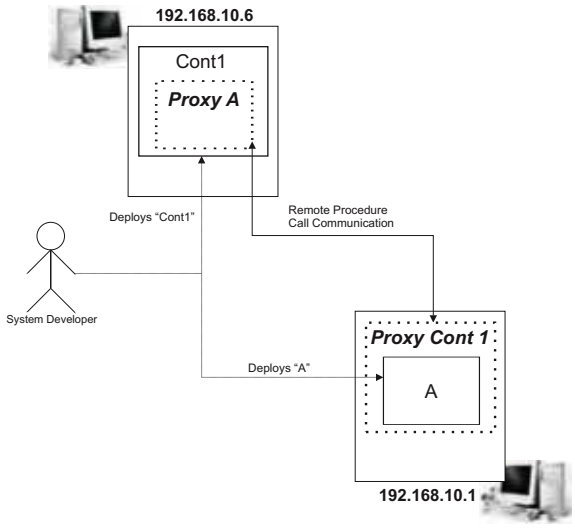


Fig. 4. Deploy and interaction of distribution mechanism.

Each instance of `ProxyFunctionalComponent` and also of the `ProxyContainer` class automatically register itself into an instance of the `Register` class. Such operation is necessary because the `Register` class is used to find proxy instances from other hosts.

In the distributed hierarchy, the component model exchanges service requests and event announcements between two computers in a transparent way. In order to implement the network communication, the JCF distribution mechanism relies on the Apache [8] implementation of the XML-RPC specification [9].

III. Example Application

The e-commerce application is a proof concept of our enterprise mechanisms. Such application provides a list of products to be purchased. In order to buy items, the user might select them. After confirming the operation, the system shows the total price of the selected items to be bought (Figure 5). When the user decides to buy something, the system invokes the *buy* service implemented in an CMS based hierarchy. This service withdraws the needed money from the user's account of a bank and deposits it into the system's account of another bank. By the presence of the transaction mechanism (described in Section II-B), the application executes the *withdraw* and *deposit* services atomically. In addition, our mechanism guarantees hardware crashes by logging the operations done by the system.

Fig. 5. Purchasing a list of items.

As illustrated in Figure 6, for security reasons, each bank is responsible for developing and maintaining both *withdraw* and *deposit* operations. The communication between the banks and the e-commerce application occurs through the distribution described in Section II-C.

This way, the application must trust in each bank components. Beyond the components, each bank must maintain a encrypted password file as illustrated in Figure 6. This password is used by the security mechanism, which is demonstrated in Section II-A.

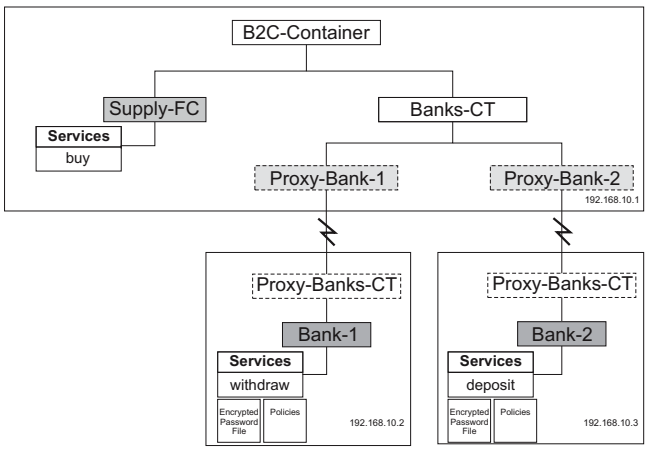


Fig. 6. E-Commerce component hierarchy.

IV. Related Works

There are two main component models used for developing enterprise applications currently, Enterprise JavaBeans (EJB) in the J2EE platform and Microsoft's .NET. The main reason is the support to the following required services in software development for enterprise applications: security, distribution, transaction, web server, etc. These features are essential and must be present into almost all applications, mainly enterprise ones.

Moreover, enterprises demand on-the-fly changes of running applications, because the downtime of their software directly causes big looses. This demand includes a new feature: The dynamic unanticipated software evolution. However neither, EJB nor .NET provide native support for this feature. This feature might be implemented in these component models, but it is a difficult task because the design of them has a high coupling among components.

There are some works which could be used to add dynamic unanticipated evolution support to these models. One of them have created a new class loader type for use in J2EE platform. This class loader might minimize the barrier between two or more class loaders [10]. Another work [11] proposes a new way to load DLL libraries in .NET.

V. Final Remarks

In this paper we presented a component based framework for developing enterprise applications supporting dynamic unanticipated evolution. Such a framework is an extension of the COMPOR Java Component Framework, which implements a component model called CMS that promotes software evolution even at runtime.

We described the framework design that is based on design patterns and aspect-oriented programming. The framework includes support to security, distribution and transaction management, but still maintaining the CMS simplicity. We describe also an e-commerce case study of the application of our approach.

As future work, we plan to develop other features of enterprise applications as extensions of JCF, such as load balancing, persistence, logging, and integration to legacy systems. Also, we are working on applying the proposed framework to develop a web-based e-commerce application.

References

[1] G. Kniesel, J. Noppen, T. Mens, and J. Buckley, "1st Int. Workshop on Unanticipated Software Evolution," in *ECOOP Workshop Reader*, ser. LNCS, vol. 2548. Springer Verlag, 2002.

[2] SUN, "Sun developer network (sdn)," July 2007, http://java.sun.com/javaee/.

[3] Microsoft, ".net framework developer center," August 2007, http://msdn.microsoft.com/netframework/.

[4] H. Almeida, G. Ferreira, E. Loureiro, A. Perkusich, and E. Costa, "A Component Model to Support Dynamic Unanticipated Software Evolution," in *Proceedings of International Conference on Software Engineering and Knowledge Engineering*, vol. 18, San Francisco, USA, 2006, pp. 262–267.

[5] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold, "An Overview of AspectJ," in *ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming*. London, UK: Springer-Verlag, 2001, pp. 327–353.

[6] M. Little, J. Maron, and G. Pavlik, *Java Transaction Processing*, 1st ed. Prentice Hall PTR, 2004.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, 1995.

[8] Apache, "Apache software foundation," August 2007, http://www.apache.org/.

[9] XML-RPC, "Xml-rpc home page," August 2007, http://www.xmlrpc.com/.

[10] Y. Sato and S. Chiba, "Negligent class loaders for software evolution." in *RAM-SE*, 2004, pp. 53–58.

[11] S. Eisenbach, V. Jurisic, and C. Sadler, "Managing the evolution of .net programs." [Online]. Available: citeseer.ist.psu.edu/728246.html

Privacy-Preserving Classification of Data Streams

Ching-Ming Chao
Department of Computer and Information Science
Soochow University, Taipei, Taiwan

Abstract

Data mining is the extraction of valuable knowledge from large amounts of data. Due to the emergence of data streams, data streams mining has recently become an important research issue. On the other hand, data mining can cause a great threat to data privacy. Privacy-preserving data mining hence has also been widely studied. However, to the best of our knowledge, there is no research that studies the privacy preservation issue of data streams mining. In this paper, therefore, we propose a method for privacy-preserving classification of data streams, called the PCDS method, which extends the process of data streams classification to achieve privacy preservation. The PCDS method is divided into two stages. The first stage uses the data splitting and perturbation algorithm to perturb data streams. The second stage uses the weighted average sliding window algorithm to mine perturbed data streams. Experimental results show our method not only can preserve data privacy but also can mine data streams accurately.

1. Introduction

Data mining is the extraction of valuable knowledge from large amounts of data. Recently, data streams have emerged as a new type of data, which have the following characteristics [1]: (1) Data has timing preference; (2) Data distribution changes constantly with time; (3) The amount of data is enormous;(4) Data flows in and out with fast speed; (5) Immediate response is required. These characteristics create a great challenge to data mining. Traditional data mining algorithms are not suitable for data streams. Hence, there have been many studies proposing efficient data streams mining algorithms. For instance, Domingos and Hulten [2] proposed the VFDT algorithm to build decision trees for data streams.

Data mining can cause a great threat to data privacy. To preserve data privacy, privacy-preserving data mining has been studied and many techniques have been proposed. Verykios et al. [3] classified privacy-preserving data mining techniques based on five dimensions, which are data distribution, data modification, data mining algorithms, data or rule hiding, and privacy preservation, respectively. Existing privacy-preserving data mining techniques are designed for traditional databases and are not suitable for data streams.

Privacy-preserving data streams mining is a very important issue. However, to the best of our knowledge, this issue has not been studied in the literature. In this paper, therefore, we propose a method for privacy-preserving classification of data streams, called the PCDS method, which extends the process of data streams classification to achieve privacy preservation. The PCDS method is divided into two stages. In the first stage, the data streams preprocessing system uses the data splitting and perturbation algorithm to perturb data streams. In the second stage, the online data mining system uses the weighted average sliding window algorithm to mine perturbed data streams. Experimental results show that the PCDS method not only can preserve data privacy but also can mine data streams accurately.

The remainder of this paper is organized as follows. In Section 2 we present the PCDS method. In Section 3 we evaluate by experiment and comparison the performance of the PCDS method. Section 4 concludes this paper.

2. The PCDS Method

2.1 The Overall Process

The overall process of the PCDS method is divided into two stages, which are data streams preprocessing and data streams mining, respectively. The primary objective of the first stage, which is handled by the data streams preprocessing system (DSPS), is to perturb data streams to preserve data privacy. The primary objective of the second stage, which is handled by the online data mining system (ODMS), is to mine perturbed data streams to construct an accurate classification model.

Data streams continuously flow in DSPS and the arriving time of data is unpredictable. If DSPS processes data streams immediately upon arrival of the data, this will consume a lot of system resources. Therefore, DSPS adopts the batch processing mode to process incoming data streams. Not only system resources can be more effectively utilized, but also data mining can be more efficiently performed. Whenever accumulating a sufficient amount of data, DSPS uses the data splitting and perturbation algorithm to perturb confidential data as well as computes the error rate resulted from data perturbation. Then DSPS passes perturbed data and the error rate to ODMS.

ODMS uses the weighted average sliding window algorithm to mine perturbed data streams to construct a classification model. Because only partial data are available for data mining, ODMS utilizes the Hoeffding bounds sampling method to construct the classification model. In addition, ODMS adopts the sliding window mode to store and process received data streams. There are two reasons for adopting the sliding window model. First, the amount of data streams is enormous and hence it is impossible to store all data. Second, users are usually more interested in more recent data. When data distribution results in a significant change, ODMS reconstructs the classification model to keep it accurate.

2.2 Data Streams Preprocessing

The primary objective of the stage of data streams preprocessing is to perturb data streams to preserve data privacy. Because data streams continuously flow in DSPS and the arriving time of data is unpredictable, DSPS is unable to collect the complete data and hence cannot use traditional perturbation techniques to perturb data streams. In addition, the data distribution of data streams can be different in different time. Using traditional perturbation techniques on data streams will increase the data error and hence will produce inaccurate mining results. As a result, whenever accumulating a sufficient amount of data, DSPS uses the data splitting and perturbation (DSP) algorithm to perturb confidential data. The DSP algorithm selects non-confidential attributes as the splitting attributes to partition the dataset. After the partition is completed, each value of each confidential attribute to be perturbed is replaced by the average value of those attribute values in its partition. When there are more non-confidential attributes used as the splitting attributes, the dataset will be partitioned into smaller subsets and the distribution of data in the same partition will be more similar. Therefore, compared to existing data perturbation techniques, the DSP algorithm has higher security and less data error. Finally, DSPS passes perturbed data to ODMS.

Figure 1 shows the steps of the DSP algorithm, which are described as follows:

step 1： Input the original dataset S. The algorithm will first partition S by building a tree. Non-confidential attributes in S will be used as the splitting attributes. Initially, the tree starts as a single node containing all records in S.

step 2： This step is to select a non-confidential attribute as the splitting attribute of the current node. Compute the variance of each non-confidential attribute based on the records contained in the current node. Select the attribute, say j^*, which has the maximum variance as the splitting attribute.

step 3： This step is to determine the splitting criterion and then partition the records contained in the current node into two disjoint subsets of records. The splitting criterion is determined by finding the median (or mid-range) of the splitting attribute. Two child nodes are generated from the current node. Each child node contains a partition of the records in the current node.

step 4： This step is to complete the partition of S. Repeat steps 2 and 3 for each node generated in step 3 until a terminating condition is reached.

step 5： This step is to perturb the confidential data in S. For each confidential attribute to be perturbed, values in each partition are replaced by their average value.

step 6： Return the perturbed dataset and pass it to ODMS.

Algorithm: Data Splitting and Perturbation
Input: an original dataset *S*
Output: a perturbed dataset *S'*
Steps:

1. Let *NA* be the set of non-confidential attributes in *S*. The tree starts as a single node containing all records in *S*.
2. Compute the variance of each attribute in *NA*. Let j^* be the attribute with the maximum variance.
3. Find the median (or mid-range) of attribute j^*. Partition the dataset in the current node into two subsets (child nodes) based on the median (or mid-range).
4. Repeat steps 2 and 3 for each of the two child nodes. Stop partitioning a node when the node contains less than a pre-specified number of records or no splitting attributes are available.
5. For each confidential attribute to be perturbed, do the following.

   For a leaf t with n_t records, let $x_{t1}, \ldots, x_{tn_t}$ be the values of the confidential attribute. Perturb the data by replacing these values with their average $\overline{x_t} = (1/n_t)\sum_{k=1}^{n_t} x_{tk}$. Repeat for each leaf in the tree.
6. Return the perturbed dataset *S'*.

Figure 1. DSP algorithm

2.3 Data Streams Mining

The primary objective of the stage of data streams mining is to mine perturbed data streams to construct an accurate classification model. ODMS uses the weighted average sliding window (WASW) algorithm, which is an extension of the VFDT algorithm, to mine perturbed data streams. Figure 2 shows the steps of the WASW algorithm. Input to the algorithm is a sequence of perturbed datasets. The algorithm adopts the sliding window mode to store received datasets and assigns different weights to different datasets according to the order of arrival. Because the value of newer data is higher than that of older data, assigning larger weights to newer data can better reflect current data distribution. Because only partial data are available for data mining, the algorithm utilizes the Hoeffding bounds sampling method to efficiently construct the classification model. Each received dataset is input to the classification model to calculate its classification error rate. A threshold value of the error rate is predetermined. The algorithm calculates the weighted average error rate of the datasets in the sliding window. When the weighted average error rate exceeds the predetermined threshold value, the algorithm will reconstruct the classification model to keep the classifica-

tion model accurate.

Algorithm: Weighted Average Sliding Window
Input: a sequence of perturbed datasets
Output: a classification model *DT*
Steps:

1. Let *W* be a sliding window of size *w*. Let *Hb()* be a split evaluation function. Let ρ be a threshold of the error rate.
2. Use *Hb()* to construct a classification model *DT*.
3. Store next dataset in *W* and calculate the error rate μ of applying *DT* on this dataset.
4. Repeat step 3 until *W* is full.
5. Calculate the weighted average error rate $\overline{\mu}$ in *W*

$$\overline{\mu} = (1/w)\sum_{i=1}^{w} \mu_i \delta_i$$

where δ_i's are time weights of datasets.

6. If $\overline{\mu} > \rho$, reconstruct a classification model *DT*.
7. Remove the oldest dataset from *W*. Store next dataset in *W* and calculate the error rate μ. Goto step 5.
8. Return the classification model *DT*.

Figure 2. WASW algorithm

3. Performance Evaluation

In this section, we evaluate by experiment the performance of the PCDS method. For data streams preprocessing, we compare the security and data error between the DSP algorithm and four existing data perturbation algorithms SAN, MN, UMA, and MMA. For data streams mining, we compare the accuracy between the WASW algorithm and the VFDT algorithm. Experimental data consist of five datasets, four of which are real world datasets and one of which is a virtual dataset generated by the synthetic data generator developed by the IBM Almaden Research Center.

3.1 Security Measurement

We use the average squared distance (ASD) and the distance-based record linkage (DBRL) between the original data and the perturbed data to measure the security of the DSP algorithm.

$$ASD = \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i)^2 \quad (1)$$

$$DBRL = \sum_{i=1}^{n}\left(\frac{x_i - \overline{x}}{\sigma(x_i)} - \frac{y_i - \overline{y}}{\sigma(y_i)}\right)^2 \quad (2)$$

x_i's are the original confidential values; y_i's are the perturbed values; N is the number of data records; $\overline{x}$ is the mean of x_i's; $\overline{y}$ is the mean of y_i's; $\sigma(x_i)$ is the standard deviation of x_i's; $\sigma(y_i)$ is the standard deviation of y_i's. ASD uses the space distance formula to measure the difference between the original data and the perturbed data. In addition to calculating the distance between two collections of data, DBRL also takes the standard deviation into account. Therefore, it can measure the variance level between the original data and the perturbed data.

Figure 3 shows the comparison of ASD measurement among the DSP algorithm and four other data perturbation algorithms using five different datasets. In all five datasets, the DSP algorithm has higher ASD values than other algorithms; therefore, it has higher security. Notice that the ASD values in the fifth dataset are lower than their corresponding ASD values in other four datasets. It is because there are less numeric attributes that can be used to perturb data in the fifth dataset. From this we can see that, in the process of perturbation, the number of numeric attributes is an important criterion to determine the risk level of data leakage. When there are more numeric attributes, data will be perturbed more seriously; therefore, the risk of data leakage will be lower.

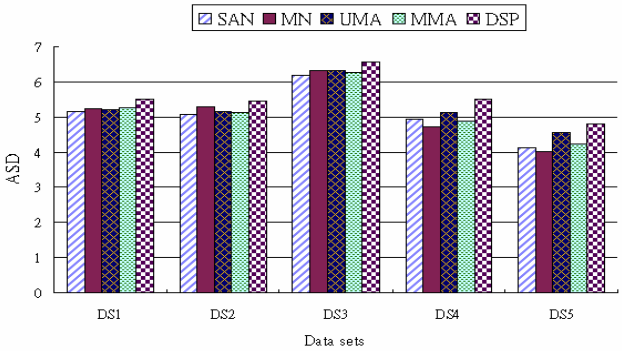


Figure 3. Comparison of ASD measurement

Figure 4 shows the comparison of DBRL measurement among the DSP algorithm and four other data perturbation algorithms using five different datasets. In all five datasets, the DSP algorithm has lower DBRL values than other algorithms, which means that the correlation between the original data and the perturbed data is lower for the DSP algorithm. Therefore, it has a lower chance to infer the original data from the data perturbed by the DSP algorithm and so the DSP algorithm has higher security.

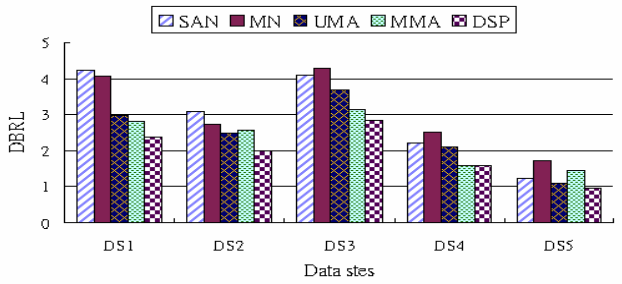


Figure 4. Comparison of DBRL measurement

3.2 Data Error Measurement

In addition to security, we also consider the data error of the mining results between the perturbed data and the original data. We use the bias in mean (BIM) and the bias in standard deviation (BISD) between the original data and the perturbed data to measure the data error of the DSP algorithm.

$$BIM = \left(\frac{\overline{Y} - \overline{X}}{\overline{X}}\right) \quad (3)$$

$$BISD = \left(\frac{S_Y - S_X}{S_X}\right) \quad (4)$$

$\overline{X}$ is the mean of the original data; $\overline{Y}$ is the mean of the perturbed data; S_X is the standard deviation of the original data; S_Y is the standard deviation of the perturbed data. BIM calculates the difference of mean between the original data and the perturbed data to measure the data error. BISD calculates the difference of variance to measure the data error. Figure 5 and Figure 6 show the comparison of BIM measurement and BISD measurement, respectively. The DSP algorithm has lower BIM and BISD values than other algorithms in most cases. Therefore, the DSP algorithm has less data error.

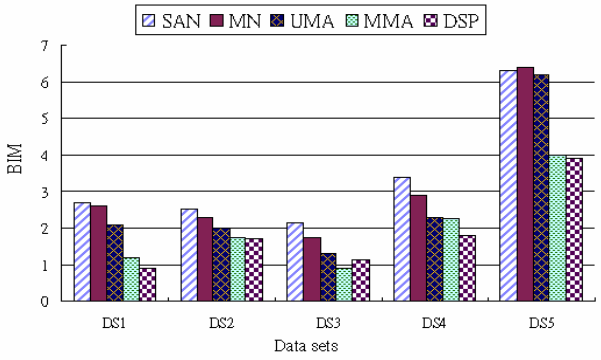


Figure 5. Comparison of BIM measurement

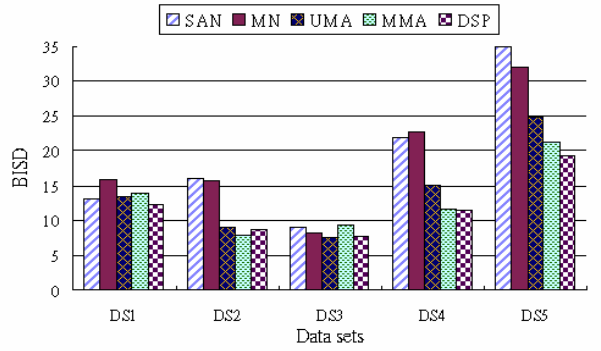


Figure 6. Comparison of BISD measurement

3.3 Accuracy Measurement

We compare the error rate of mining perturbed data between the WASW algorithm and the VFDT algorithm. The threshold value of the error rate in the WASW algorithm is set to 15%. Figure 7 shows experimental results on various data volumes. The initial error rate of the VFDT algorithm is 10%. Along with continuous arrival of the data stream, the error rate will increase constantly. On the other hand, although the initial error rate is 12%, the WASW algorithm will reconstruct the classification model to reduce the error rate when the error rate exceeds the predetermined threshold value. Therefore, the WASW algorithm can adjust to current data distribution to maintain the accuracy of the classification model.

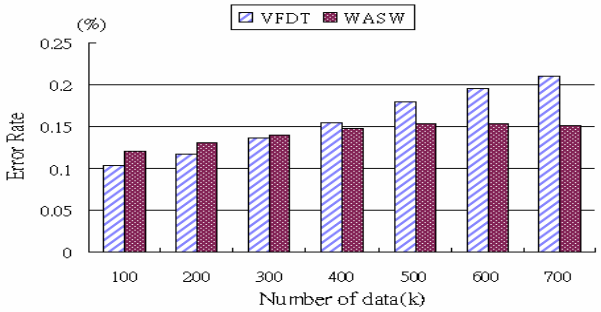


Figure 7. Comparison of the error rate

4. Conclusion

This paper proposes a method for privacy-preserving classification of data streams, which consists of two stages. The first stage uses the DSP algorithm to perturb data streams. Experimental results show that the DSP algorithm has higher security and less data error. The second stage uses the WASW algorithm to mine perturbed data streams. Experiment results show that the WASW algorithm has higher accuracy. Therefore, our method can not only preserve data privacy but also mine data streams accurately.

Acknowledgment

The author would like to express his appreciation for the financial support from the National Science Council of Republic of China under Project No. NSC 96-2221-E-031-001-MY2.

References

[1] L. Golab and M. T. Ozsu, "Issues in Data Stream Management," *ACM SIGMOD Record*, Vol. 32, No. 2, pp. 5-14, June 2003.

[2] P. Domingos and G. Hulten, "Mining High-Speed Data Streams," in *Proceedings of the 6th ACM International Conference on Knowledge Discovery and Data Mining*, Boston, MA, pp. 71-80, 2000.

[3] V. S. Verykios, K. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis, "State-of-the-art in Privacy Preserving Data Mining," *ACM SIGMOD Record*, Vol. 33, No. 1, pp. 50-57, March 2004.

Comparing the Use of Traditional and Associative Classifiers towards Personalized Recommendations

Joel Pinho Lucas, Saddys Segrera, María N. Moreno
Department of Computer Science and Automatics, University of Salamanca
Plaza de la Merced s/n, 37008, Salamanca, Spain
{ joelpl,saddys,mmg}@usal.es

Abstract

Nowadays, there is a diversity of methods, including data mining techniques, available to be used in recommender systems. However, such systems still present numerous limitations. An alternative data mining technique is the classification based on association, which combines concepts from classification and association. In this way, association rules perform a predictive role. These techniques, not very extended, have not been applied in the recommender systems area. In order to provide recommendations more effectively, in this work we suggest that associative classifiers can be also used in these type of systems. Therefore, a revision about the recommender systems and the methods of associative classification is presented. In addition, a case study is described, in which traditional and associative classifiers are evaluated and compared using data from recommender systems. The results showed that both type of classifiers can be applied effectively in such systems, nevertheless punctual characteristics on data determine which is most effective in each scenario.

1. Introduction

It has been estimated that the amount of information in the world doubles every 20 months [8]. Additionally, nowadays it is technologically feasible to store huge volumes of data with, more and more, lower costs. However, it results in an "information explosion" where there is a lot of useless data in which it is very difficult to find valuable information.

In e-commerce systems such "information explosion" is reflected by loads of products available for sale. In this way, users would probably have difficulty in choosing the products they prefer and, consequently, have difficulty purchasing them. Due to such facts and to a more and more competitive industry, these systems need to personalize the presentation of their products to the consumers.

A way to reach such personalization is by means of the "recommender systems", which are being used by an ever-increasing number of E-commerce sites in order to help consumers to find products to purchase [17]. There are two types of errors that these systems can present: false negative and false positive. The first one consists of products that were not recommended, though the consumer would like them. The second one consists of recommended products, though the consumer does not like them. According to Sarwar et al. [16], false positives are more critical because they will lead to angry consumers.

Taking into account that data mining techniques are applied for identifying patterns within data sets, according to Cheung et al. [4] these techniques can be successfully applied for recommender systems, however they need to be extended to deal with common issues for such systems. The induction of association rules is a data mining technique widely applied in decision making processes, which was first introduced by Agrawal et al. [1] in the context of market basket analysis.

Despite being a non-supervised learning method, association rules induction can also be applied to perform classification tasks. In this work we suggest the development of recommendation models using association rules in a prediction perspective, which are usually referred as associative classifiers. In order to analyse the behavior of such classifiers on a recommender systems' data, we accomplished a case study using two recommender systems databases.

The key novelty of this work is the use of association rules, for classification tasks in recommender systems. Moreover, in order to identify in which situations each algorithm may be used more effectively, we depict some features and issues of particular algorithms that are intrinsically related to data characteristics.

In the next section we describe general features of recommender systems, where the main drawbacks they present are highlighted. The use of association rules for classification problems are described in section 2. In section 3 we highlight some concepts related to recommender systems. In section 4, we describe the case study accomplished on two databases. Finally, in section 5, we describe an evaluation analysis about the case study.

2. Recommended Systems

According to Cheung et al. [4] and Lee et al. [10], the methods implemented in recommender systems can be divided into two main classes: collaborative filtering and content-based methods. Content-based methods compare text documents to user profiles, where web objects are recommended to a user based on those he has been interested in the past [10]. Hence, recommender systems that use such type of methods do not take into account information acquired by other users. On the other hand, in collaborative filtering methods the recommendation process is based on products' opinions collected from other users [5]. According to Sarwar et al. [15], the collaborative filtering approach was originally based on nearest neighbor algorithms, which recommends products to a target user according to the opinions of users who have similar purchase historical. Thus, the recommended products will be the ones which users with similar interests have been liked.

Breese et al. [2] classified collaborative filtering methods into two groups: memory-based methods, which are also referred as user-based methods, and model-based methods, which are also referred as item-based methods. In memory-based methods the nearest neighbors of a target user is found by matching the opinions of such user to the opinions of all system's users. On the other hand, model-based methods build a predictive model by means of a training set which comprises opinions acquired from a small portion of the system's users. Such methods have been developed more recently in order to avoid the sparsity problem, which usually arises when memory-based methods are employed, because e-commerce systems generally offer millions of products for sale, so that it is not feasible to obtain opinions about all of them [16]. As a result, current recommender systems typically do not employ merely memory-based methods.

Likewise, content-based recommendation methods usually are not employed solely, because they are not effective due to the lack of mechanisms to extract Web objects features. However, such methods are commonly employed in conjunction with collaborative filtering methods.

Taking into account model-based collaborative filtering, machine learning techniques are the most employed methods. Several machine learning techniques that are employed to solve data mining problems are also employed in recommender systems. Furthermore, nowadays numerous systems employ the agents' technology combined to those techniques [3]. The use of agents in these systems is basically due to their autonomy, learning capability and the possibility of working in cooperation [14]. In the next sub-section we will state some of the most critical drawbacks presented by methods employed in recommender systems.

2.1 Drawbacks

The most critical drawback such methods presents is probably associated to data sparsity, due to the large number of items that current recommender systems usually present. According to Sarwar et. al. [15], users of e-commerce systems are able to purchase, in general, only 1% of the products available by the system. This constraint is more problematic for memory-based collaborative methods, because it may not be feasible to obtain enough ratings from users of a system. Model-based collaborative filtering methods can minimize drawbacks originated due to sparsity.

Another drawback originated from the large number of items available in recommender systems is scalability. Such drawback may turn into the major concern to the system performance, because the process of searching the nearest neighborhood, for example, may be unfeasible for systems that encompass huge data bases. The performance is a key feature in recommender systems, because these systems need to provide their users fast feedback. Generally, scalability is not a drawback for model-based methods, because in such methods, differently from others, main data processing, the induction of the predictive model, usually is not performed at run time.

Despite of the drawbacks mentioned above been able to be brightened up by means of model-based collaborative filtering methods, there are some drawbacks that these methods can not solve. The early rater problem [5] [7] is an example of drawback that may occur in all type of collaborative filtering methods. Such problem is related to the restraint of having few opinions on which to base the predictions.

Conversely, there are drawbacks, such as the grey sheep problem [5], that occur only in collaborative filtering methods. The grey sheep problem refers to the users who have opinions that do not consistently agree or disagree with any group of users. As a consequence, such users would not receive recommendations. However, such problem does not occur in content-based methods, because such methods do not consider opinions acquired from other system's users in order to make recommendations. According to Condliff et. al. [7], since a content-based system does not consider the social background of its users, the system is limited to recommend just items that are similar to those that a user has liked in the past.

3. Classification Based on Association

As stated before, association rules induction algorithms can be employed to build recommendation models such as the one in [10]. Association rules were first introduced by Agrawal et al. [1] aiming at discovering consuming patterns in retail databases. Thus, the task of discovering association

rules in retail data was termed as "market basket analysis". The representation of an association rule may be declared as $A \rightarrow B$, where A and B are item sets. Such representation states that, in a transaction, the occurrence of all items from "A" (antecedent side of the rule) results in the occurrence of items belonging to "B" (consequent side of the rule), such as $A \subseteq I$ and $B \subseteq I$, where "I" is an item set. An association rule describes an association relation between item sets that occurs together on transactions of a data set. Thus, association is not considered as a prediction task, because it aims at describing data.

On the other hand, a classification method is seen as a prediction task, because it aims at predicting the value of an attribute (label) in a data set. The joining of concepts from classification and association [12] is an alternative approach for performing classification tasks, where association rules are employed as a classification method. Seeing that association models are commonly more effective than classification models, a crucial matter that encourages the use of association rules in classification is the high computational cost that current classification methods present. Several works [11] [12] [18] [21] verified that classification based on association methods presents higher accuracy than traditional classification methods. Differing from association rules, the decision trees, for example, do not consider simultaneous correspondences occurring on different attribute values. Moreover, human beings can easier comprehend an output provided by association rule algorithms than an output provided by usual classification techniques, such as artificial neural networks [16].

According to Thabtah et al. [18], a few accurate and effective classifiers based on associative classification have been presented recently, such as CBA (Classification Based in Association) [12], CPAR (Classification based on Predictive Association Rules) [21], MCAR (Multi-class Classification based on Association Rule) [18], CMAR (Classification based on Multiple class-Association Rules) [11] and TFPC (Total from Partial Classification) [6]. Taking into account that for classification rule mining there is one and only one predetermined target, while for association rule mining the target of discovery is not pre-determined [12], it is necessary to constrain the rules' consequent terms to encompass only one attribute. This way, the consequent term of an association rule will represents the target, or class, attribute. Therefore, such rule can play a prediction role in a given system: in order to classify an item, the rule's properties are matched to every rule's antecedents and the attribute value of the consequent term (from one or more selected rules) will be the predicted class. Generally, the classification model is presented as an ordered list of rules, based on a rule ordering scheme [19].

In the CBA algorithm, for example, the rules are ordered by means of the confidence measure and it uses only one rule for performing classification. However, in this case some scenario in which could exist multiples rules with similar confidence measures may occur and, at the same time, with greatly different support measures. Hence, a rule A with much higher confidence than a rule B could be the one chosen for classification even if B had a much higher support [11]. The MCAR algorithm solves such drawback by means of an approach that considers, in addition to the confidence, the rules' support. The CMAR algorithm has a fine approach for selecting association rules for classification, instead of using just one rule it makes use of all rules that match the case to be classified. If the consequent term of all selected rules is the same, the predicted class will obviously be the value of the rules' consequent term. Though, in a different scenario, rules are divided in groups according to the consequent terms' values. The value chosen for classification is acquired through the group in which its elements hold the highest correlation value according to the weighted $\chi 2$ measure. Similarly to CMAR, the CPAR algorithm also divides rules in groups, though, instead of using all rules that match to the object to be predicted, it uses the "k" best rules that represent each class. Afterwards, the algorithm chooses a group, by means of the Laplace Accuracy measure, that will be the one used for classification.

The drawbacks presented by association rules induction algorithms are, in general, the same ones of classification based on association algorithms. A critical drawback of these algorithms is due to those rules that have few attributes. Seeing that such rules expresses narrow information, an object which has few attributes would be ineffectively classified. Another critical drawback is due to the large number of rules that algorithms commonly produce [16], as a consequence, much of them do not supply relevant information or are contradictory. Such drawback is a critical issue related to associative classifiers, because the performance of the algorithm may be affected when retrieving, storing, pruning and sorting a large number of rules [11]. The CMAR algorithm presents tries to solve such drawback by implementing a FP-Tree data structure to store the association rules' frequent itemsets.

4. Case Study

In this section we describe a case study accomplished on two databases: MovieLens and Book Crossing. The first consists of ratings of movies made by MovieLens users in 2000, which is a recommender system based on the GroupLens technology. Such database is freely available for research purposes on the GroupLens Web page [9]. The second consists of book ratings gathered by Ziegler et. al [22] from the Book-Crossing community, whose users exchange books and their experiences all around the world.

For both databases the WEKA [20] tool was used to perform data transformation and pre-processing. The next subsections detail both databases and how they were used.

4.1 MovieLens Data

Initially, the MovieLens dataset contained approximately 100,000 ratings for 1,682 movies made by 943 users, where we integrated the data related to users and movies into one file, which was the input provided for the algorithms analyzed in this case study.

However, before supplying such input we changed the rating attribute in order to have only two values: "Not recommended" (score 1 or 2) and "Recommended" (score 3, 4 or 5). The first one refers to an item the user may like and the second refers to the opposite case. Such changes were performed to simplify classification, because the main aim in a recommendation task is to determine if an item should be offered to the user. Taking into account users' data, we used the following attributes: gender, age and occupation. The age attribute was discretized in five age ranges. The user's occupation attribute is a nominal variable with 21 distinct values.

Taking into account movies' data, the file provided by MovieLens originally contained 19 binary attributes related to movie genres. An instance with value 1 expressed that the movie belongs to a specific gender and 0 otherwise. The association model's consistency would be compromised if 19, among the 23 attributes on the dataset, were binaries. Thus, these 19 binary attributes were reduced to just one attribute representing the movie genre's name. However, since some movies may belong to different film genres, we only used the records containing ratings about movies with just one genre. Afterwards, 7 film genres were not considered in this study. Hence, association rules generated by the model could express relationships between user profiles' characteristics and film genre features.

After data pre-processing and transformation, 14,587 records were remained in the input file for the algorithms used in this study. For experiments made on classification based on association algorithms we defined support and confidence threshold values of 20% and 75% respectively.

4.2 BookCrossing Data

Initially, the Book Crossing data contained 1,149,780 ratings about 271,379 books provided by 278,858 users. However, such ratings include explicit (an assigned mark from 1 to 10) and implicit (written reviews) ratings. Thus, the implicit ratings were not considered for this study and the dataset remained with 433,671 records.

In order to simplify classification, the rating attribute was modified in the same way of MovieLens was: "Not recommended" (score from 1 or 6) and "Recommended" (score from 7 to 10). For books' data, we used two attributes from the dataset: publication Year and Author. The first was discretized in five ranges. The Author attribute was also modified, because at first it encompassed 48,234 distinct values. Thus, the dataset was reduced in order to this attribute encompasses only 40 distinct values (the ones that appear on more records).

Taking into account users' data, we also used two attributes: Age and Place where the user inhabits. The first was discretized in nine age ranges. The Place attribute originally contained the name of the city, the state or province, and the name of the country. However, this way such attribute presented 12,952 distinct values. Therefore, we changed this attribute in order to encompass only 40 distinct values. For that reason and seeing that 75% of the places were from USA, we divided the dataset, based on this attribute, in two: places grouped by states of USA and places grouped by countries excepting USA. Afterwards, the first dataset (states of USA) remained with 25,523 records and the second one (countries) remained with 8,926 records.

In order to try the algorithms' accuracy facing a smaller range of distinct values we also used more two datasets derived from those ones mentioned before. Thus, we copied both datasets and kept only 10 distinct values (the most frequent) for author and Country/State attributes. This way, we obtained two more datasets that contain 6,270 records (on the dataset of states of USA) and 3,238 records (on the dataset of countries).

To perform the experiments on the classification based on association algorithms we defined support and confidence threshold values of 20% and 80%. However, for the datasets containing 10 distinct values, we reduced the support to 10% due to its reduced number of records.

4.3 Results

The classifiers analyzed in this case study were the following ones: C4.5, BayesNet, CBA, CPAR and CMAR. The first two were run through WEKA and the other three were obtained from the LUKS-KDD repository [13]. In all experiments we used 50% of the dataset as a training set and 50% as a test set.

The main objective was to compare the algorithms accuracy using data gathered from recommender systems. For CBA, CPAR and CMAR we also analysed the number of rules generated for building the classification models. In total, five datasets were analyzed: one obtained from MovieLens database and four through Book Crossing database (the dataset of states of USA, of world countries excepting USA and the same two but with 10 distinct values for author and Country/State attributes). On table 1 we show the results obtained after running the algorithms mentioned above. Each line depicts the accuracy obtained on each dataset and, for the associative classifiers, it is also depicted the number of rules available on classification models.

Table 1. Comparison of Classifiers.

Data	C4.5	BayesNet	CBA	CPAR	CMAR
MovieLens	83.45%	81.16%	78.35% – 13R	74.07% – 65R	85.16% – 11R
BCrossing World	80.30%	78.33%	78.96% – 2R	73.25% – 127R	0.0% – 0R
BCrossing World 10	80.23%	81.09%	82.58 – 4R	79.86% – 28R	9.29% – 1R
BCrossing USA	80.33%	80.23%	80.49% – 4R	78.15% – 264R	9.68% – 1R
BCrossing USA 10	80.89%	80.82%	73.11% – 10R	76.71% – 42R	24.66% – 3R

Results revealed that associative classifiers reached similar accuracy, excepting CMAR on Book Crossing data, to traditional classifiers (surpervised learning). Actually, in some cases associative classifiers reached higher accuracy. Despite of being the first method of classification based on association, the CBA algorithm reached the highest accuraccy on two of the four datasets of Book Crossing. On MovieLens data, the CMAR reached the highest accuracy, which was the best result obtained over all experiments.

Since rules provided by the associative classifiers hold a high confidence value (at least 80%), the rules used for building the classification models are reliable. The ninth rule generated by CMAR on MovieLens data is an example of this kind of rule: "age=[25-34] & genre='drama' => rating='yes'". Such rule states that, if a user is older than 25 years and younger than 34 years old, he will probably rate positively a drama movie.

5. Empirical Analyses

Despite of presenting the highest accuracy over all experiments (85.16% on MovieLens data), CMAR did not present satisfactory results on Book Crossing data. MovieLens and Book Crossing data basically differ on the number of distinct values of their attributes. MovieLens has only two distinct values on the Genre attribute, for example, and the other attributes have, in general, less distinct values than MovieLens datasets when the ratio of records/number of distinct values is taken into consideration. In addition, on the datasets with ten distinct values CMAR presented a slight improvement on the classification accuracy. This may be justified by the data structure it employs, which is a FP-Tree (Frequent Pattern Tree). This structure stores frequent itemsets in a compact way in which common relations between itemsets are explored. This way, for Book Crossing data, items stored in the FP-Tree were not frequent enough to form rules. Due to such outcomes, we argue that CMAR is more effective on datasets that encompass attributes with less distinct values.

On the other hand, the CBA algorithm presented good results on Book Crossing data. CBA uses the same data structure and foundations of the trivial Apriori algorithm, which does not encompass a compact structure for storing items. Under these circumstances CBA provided enough rules to build effective classifiers. However, there was a loss of accuracy in two cases: for the dataset of world countries (excepting USA) and for the dataset of states of USA with 10 distinct values. Such losses are reasonable due to the diversification of characteristics that readers from different world countries (from four continents) may have, which probably are much more substantial than people from just one country. Such diversification is even more crucial to the CBA association rule induction, because before classifing it has to obtain association rules based on relationships over attributes. For the dataset composed by states of USA with ten distinct values, it is even more difficult to identify relationships, because readers from the remained ten states probably present quite similar characters. On the other hand, for the dataset of world countries with ten distinct values, readers' characters are more dissimilar and then the CBA accuracy was improved.

At last, the CPAR algorithm also presented acceptable results, even though its accuracy was slightly lower than ones of other classifiers. Such algorithm is more effective for scenarios of very large datasets where processing time may be a critical issue, because the classifier construction and the rules induction are made in just one processing step. Moreover, results showed that the algorithm provides much more rules than CBA and CMAR. It may be useful for scenarios where data description is also an analyses' aim.

6. Conclusions

In this work we have shown, by means of the case study described in section 4, associative classifiers can be used in recommender systems effectively and can also improve recommendations consistency. Actually, both traditional and associative classifiers can be applied effectively, nevertheless punctual characteristics on data determine which is most effective in each scenario. Through this work we intend to decrease errors on recommender systems, because such systems still encompass several shortcomings.

According to the conclusions of some works, including this one, the accuracy of classification methods has a straight correlation to data's attributes characteristics. Thus, before applying a classifier it is essential to analyse data attribute. Seeing that, none of the works that proposed classification based on association methods had investigated what and

how datasets' features affect classification accuracy, on the previous session we described circumstances that each algorithm would be more effective and likely to be used.

Associative classifiers provide fast and comprehensible learning models, differing from the majority of traditional classifiers. Another major achievement of associative classifiers in this work was the few false positives that would be presented in recommendations. Since we established a high threshold value for confidence (80%), rules provided for the learning model would not be likely to classify an item as "Recommended" if it was interesting to a user whose characteristics (attributes) did not match the items on rules' antecedent terms. On the other hand, false negatives might occur easier, because certainly all rules do not encompass information about all datasets' relations. However, if an item of interest to a user was classified as "Not Recommended", it would not be a critical error.

In order to classification based on association methods be applied (as well as general data mining techniques) even more effectively in recommender systems, in future works we will try to bring up more personalization for some associative classifiers for recommender systems data.

Acknowledgments

This paper is part of the research project SA064A07, supported by the Spanish *Junta de Castilla y Leon*.

7. References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In the ACM SIGMOD International Conference on Management of Data, pp207–216, Washington, USA, 1993.

[2] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. pages 43–52, 1998.

[3] M. Chau, D. Zeng, H. Chen, M. Huang, and D. Hendriawan. Design and evaluation of a multi-agent collaborative web mining system. Decision Support Syst., 35(1):167–183, 2003.

[4] K.-W. Cheung, J. T. Kwok, M. H. Law, and K.-C. Tsui. Mining customer product ratings for personalized marketing. Decision Support Syst.,35(2):231–243, 2003.

[5] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin. Combining content-based and collaborative filters in an online newspaper, 1999.

[6] F. Coenen, P. H. Leng, and L. Zhang. Threshold tuning for improved classification association rule mining. In PAKDD, pages 216–225, 2005.

[7] M. Condliff. Bayesian mixed-effects models for recommender systems, 1999.

[8] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases – an overview. Ai Magazine, 13:57–70, 1992.

[9] GroupLens. GroupLens Research Group. University of Minnesota. http://www.grouplens.org/.

[10] C.-H. Lee, Y.-H. Kim, and P.-K. Rhee. Web personalization expert with combining collaborative filtering and association rule mining technique. Expert Systems and Applications., 21(3):131–137, 2001.

[11] W. Li, J. Han, and J. Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. In ICDM, pages 369–376, 2001.

[12] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In Knowledge Discovery and Data Mining, pages 80–86, 1998.

[13] LUKS-KDD. The Lucs-KDD Software Library. University of Liverpool. http://www.csc.liv.ac.uk/ frans/KDD/Software/.

[14] M. N. Moreno, F. J. García, M. J. Polo, and V. F. Lopez. Using Association Analysis of Web Data in Recommender Systems, volume 3182/2004, chapter E-Commerce and Web Technologies, p11–20. Springer, 2004.

[15] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In World Wide Web, pages 285–295, 2001.

[16] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In ACM Conference on Electronic Commerce, 2000.

[17] J. Schafer and J. Konstan. E-commerce recommendation applications. Data Mining and Knowledge Discovery, 2001.

[18] F. Thabtah, P. Cowling, and Y. Peng. MCAR: multi-class classification based on association rule. In Proceedings of the International Conference on Computer Systems and Applications, Washington, USA, 2005. IEEE.

[19] Y. Wang, Q. Xin, and F. Coenen. A novel rule ordering approach in classification association rule mining. In 5th International Conference on Machine Learning and Data Ming, pages 339–348. Springer LNAI 4571, 2007.

[20] WEKA. Waikato Environment for Knowledge Analisys: Machine Learning Software in Java. University of Waikato. http://www.cs.waikato.ac.nz/ml/weka.

[21] X. Yin and J. Han. Cpar: Classification based on predictive association rules. In SIAM International Conference on Data Mining 2003, pages 331–335, 2003.

[22] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In 14th International World Wide Web Conference, 2005.

Discovering Meaningful Clusters from Mining Software Engineering Literature

Yan Wu, Harvey Siy, Li Fan

College of Information Science and Technology,
University of Nebraska at Omaha,
Omaha, Nebraska 68182, USA,
{ywu, hsiy, lfan}@mail.unomaha.edu

Abstract

Document clustering is becoming an increasingly popular technique for identifying relationships in unstructured text. In this paper, we attempt to make sense of the output of a clustering algorithm applied to software engineering research papers. We introduce a notion of cluster "stability" as a measure of the meaningfulness of a cluster. We assess its usefulness and limitations in identifying meaningful clusters. In the process, we track how important research topics may have changed from year to year.

1. Introduction

In many fast-moving fields, such as software engineering, a large number of research papers are being published in journals and conferences every year. Due to the increasing volume of papers, it is hard to effectively and efficiently search for papers relevant for a given topic of interest. And as research areas become increasingly interdisciplinary, it is much more difficult to differentiate research topics of published research papers. Important academic digital libraries such as ACM Digital Library have constructed the hierarchical categories for research areas but many research papers do not have a good fit within the available categories.

The goal of document clustering is to automatically categorize unlabelled documents into cohesive clusters to facilitate the search for relevant information. But due to the unsupervised nature of clustering algorithms, it is hard to decide the number, *k*, of clusters we need when applying clustering. In [1] the authors mentioned that by inspecting the output of the clustering algorithm, various values of *k* can be tried. However, this assumes that inspecting the output of the clustering is feasible. Especially in fields such as software engineering, it is hard to ascertain the number of research streams to be differentiated. So, we are in need of a procedure to empirically determine the number of clusters for a given set of documents and to validate the meaningfulness of the resulting clusters. Such a procedure can be used to guide future attempts to cluster research papers in this area. In this paper we attempt to make sense of the output of a clustering algorithm applied to two bodies of text, one from a set of journal articles and another from a set of conference papers. The texts were partitioned by year of publication in order to identify any presence of trends.

2. Related Work

Hierarchical clustering algorithms are described in [13]. The advantage of hierarchical clustering solutions is that they provide different views of the data at different granularities. We chose hierarchical clustering as it enables us to track the clustering process and helps to identify a suitable level of granularity that yields many meaningful clusters.

[4] tried to achieve more balanced and stable clustering by exploiting the characteristics of data objects before clustering to eliminate the effect of data ordering on clustering number. This research is focused on the elimination of the effect of data ordering to improve the quality of clustering. Authors of [5] worked on evaluating quality of a certain clustering algorithm --- Adaptive Resonance Theory (ART) neural networks by certain formulas concern the numbers of true positive, false positive and false negative in clustering results. That is a work with prior knowledge of the objects for clustering as a validation work, but we are working on an exploring work in which almost no existing information provides us clues about "correct answer".

We also found several papers related to labeled classification, which we employ to validate the resulting clusters. The work in [10] is focused on identifying descriptive, sensible clustering labels for each cluster by an approach named as Description Comes First (DCF) cluster labels, in which the processes of candidate cluster label discovery and document clustering are separated. In [12] authors employed frequently appeared phrases as final cluster description, which employed a similar approach as what we used. A follow-up work [2] showed how to avoid certain suffix tree clustering (STC) limitations from [12] and use non-contiguous phrases. A

"label-driven" clustering appeared in clustering with committees algorithm, where the semantic relationships from WordNet are used to evaluate the unambiguous concepts to which strongly associated terms are related.

Several lines of investigation also deal with detecting temporal trends. [9] performs trend analysis in order to find hot topics through controlled vocabulary terms rather than phrases based on the nature of news that smaller units could be used to identify breaking news topics within short period such as one day. Temporal Text Mining (TTM) described in [6] is used to discover and summarize the evolutionary patterns of themes in a text stream over time. Based on the discoveries in characteristic path, authors of [3] collected the paper titles from DBLP XML files to track the most popular terms used throughout time. Then they listed the emerging popular terms for each year by deleting terms that appeared in the previous two years and by this way they explained the previous discoveries.

3. Methodology

The bodies of text consists of papers from three software engineering journals, *IEEE Transactions on Software Engineering*, *ACM Transactions on Software Engineering and Methodology*, and *Empirical Software Engineering*, and the proceedings of two software engineering conferences, *International Conference on Software Engineering (ICSE)* and *ACM SIGSOFT Foundations on Software Engineering (FSE)*. We collected a total of 1157 journal articles from 1996-2007 and 675 conference papers from 1999-2007.

We partitioned the papers according to year of publication and clustered each year individually. We also clustered each set of text as a whole. We kept the analysis of journal articles and conference papers separate as the two sets are likely to show different trends owing to different timescales for publication turnaround times.

3.1 Converting Documents into Word Vectors

Before clustering, we need to convert the text into lists of word vectors for input to the clustering tool. For this, we used RapidMiner [11], an open source data mining environment which supports most frequent data mining tasks. The modular operator concept of RapidMiner allows the design of complex nested operator chains to solve a wide variety of learning problems effectively and efficiently. Rapidminer is a common text mining too and it is easy to use, its extensibility and flexibility provide us the ability to expand our work in the future. Data handling in RapidMiner is transparent to users. To vectorize text, the text plug-in for RapidMiner is recommended [11], which can be used to create word vectors from input texts in different formats (plain text, URLs and so on). To create word vectors from input texts, a list of operators are chosen and put in order. First, the texts need to be read from certain document directory so that further treatment could be executed on them (TextInput Operator). As one of the parameters, *tf-idf* is chosen as the term weighting model, in which each document can be represented as

$$tf_1\log(n/df_1),\ tf_2\log(n/df_2),\ \ldots,\ tf_m\log(n/df_m)$$

where tf_i is the frequency of the ith term in the document, df_i is the number of documents that contain the ith term and n is the total number of documents. With *tf-idf*, those words that appear frequently but in fewer documents receive higher weight. Second, the plain text is tokenized (StringTokenizer Operator). Third, stop words such as "a" and "is" are deleted (EnglishStopwordFilter Operator). The last step is stemming, which truncates words into their roots and combines words with the same root together (PorterStemmer Operator).

After the execution of the operators, an example set is produced as the result. The Metadata View of the result stores the type, name, value type, statistics and range of attributes (stemmed words), the Data View stores the data – the vector-space model representing documents.

Processing the full text of documents would result in very large word vectors that are too unwieldy to analyze. Furthermore, analyzing the full text increases the likelihood of the clustering algorithm making false associations between documents. Therefore, to facilitate the processing and reduce the scale of attribute dimensions, only the title, abstract and keywords are extracted from those papers. Although they do not contain the full contents, the title, abstract and keywords generally capture the key information in a document.

3.2 Hierarchically Clustering Documents

We applied the hierarchical clustering tool from SPSS to cluster the documents. This is an *agglomerative* hierarchical clustering tool which clusters from the bottom up, treating each single text as one cluster, and then merging similar texts together until it has merged all documents into one cluster.

To reduce the number of single-document clusters, we began collecting data on the merging process when there were 20 clusters left. We recorded the list of documents in each cluster and also which clusters were merged in

each subsequent step until there were only two clusters left to be merged. The SPSS output was saved as a spreadsheet for further processing.

3.3 Identifying Stable Clusters

To quantify the meaningfulness of the resulting clusters, we computed the *stability* of each cluster. Taking the SPSS output, we determined, for each cluster, at which merge step a cluster was formed, and at which step it got merged into another cluster. The difference between the two steps is the measure of a cluster's stability. For example, if a cluster was formed at the 1st merge step, and was merged at the 5th step, it stayed intact for 4 runs. We consider clusters that stayed intact for more than 10 runs as stable, because we have tried to consider smaller number of runs as stable clusters but it was hard to find out overall themes for those clusters.

We also calculated the overall stability of a merge step as the sum of the runs for all the clusters available at that step. In this way, we have a way to determine k, the number of clusters with the highest overall stability.

3.4 Constructing Tagclouds for Clusters

To validate the meaningfulness of the resulting clusters, we inspected the dominant words in each cluster using *tagclouds* [8]. Tagclouds provide visual presentations of a set of words in which the size and color of a word are used to represent some attribute of the word, such as its frequency or how recently it was used. We use tagclouds to visualize the dominant words in each cluster. The accumulated weight of a word w in a cluster is calculated by totaling the weights of w in each of the documents in the cluster. In the tagcloud visualization, the fonts used to display the words are proportional to their accumulated weight. An example is given in Figure 3. In this example, the words are sorted according to their accumulated weights.

Once stable clusters are identified, we inspected the tagclouds of these clusters to identify the *topics* within each cluster. This was then verified by examining the paper's original title and abstract.

3.5 Classification by Labels from Stable Clusters

With the hypothesis that the stable clusters are distinct clusters which have relatively farther distances from other clusters, a validation is performed by attempting to classify all documents in the text body into the identified stable clusters. As the starting point, papers in each year's stable clusters are collected and separately grouped and labeled together, and one comparison group is created as part of the training set. Using RapidMiner, the LibSVMLearner and ModelWriter operators are applied to train the classification model. This process takes as input the labeled groups of papers and the outputs include a wordlist and a model in which the classification criteria are stored. Then, to verify the hypothesis that the stable clusters are distinct clusters that keep relatively longer distance from other clusters and belongs to special topics, the classification model trained on one year's documents is applied onto the documents in the other years. In this way, some originally undiscovered papers are located and further evaluation is needed to prove they really belong to the special stable clusters.

4. Results

4.1 Hierarchical Clustering Results

The results from agglomerative clustering of documents per year indicate several clustering patterns. Large clusters attract additional clusters and tend to get merged first. This is not surprising since large clusters have many important words that increase the likelihood of being related to other clusters. We see this pattern appearing more in journals than in conferences. Conference clusters are more likely to consist of small clusters, without one cluster dominating all the others. We can also see this trend by examining the hierarchy trees which are derived by following the agglomeration process that successively merges a pair of clusters at each step. Journal clusters tend to produce unbalanced hierarchy trees, while conference clusters tend to produce relatively more balanced trees. These trends are illustrated by Figure 1 and Figure 2, which depict the hierarchy trees when clustering all journal and conference papers, respectively.

In these figures, the merging process is illustrated by two clusters pointing into one (merged) cluster. In the clustering based on journal articles (Figure 1), most of the initial clusters (those without any clusters pointing into them) are successively merged into a bigger cluster (starting from the one in the top left corner), resulting in an unbalanced tree. On the other hand, the clustering based on conference papers (Figure 2) did not have such a dominant cluster at the start, and the merging process appears to be more balanced. Examining the clustering hierarchy trees for individual years also produce consistent results for both journal and conference papers.

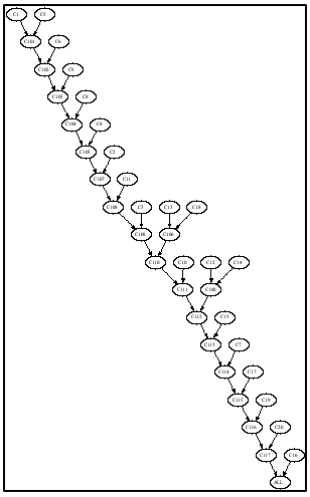


Figure 1: Clustering hierarchy tree for journal articles

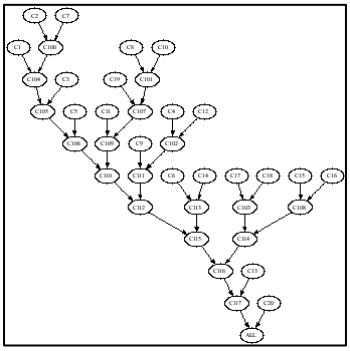


Figure 2: Clustering hierarchy tree for conference papers

4.2 Stable Clusters

For each year's clustering process, we also detected a subset of clusters which tend to be preserved for several runs of the agglomerative clustering algorithm, indicating that their contents are quite distinct from the others. We consider that a cluster is stable if it survives without getting merged through more than 10 runs. For such clusters, it is generally easier to find overarching themes based on their dominating keywords.

For example, Figure 3 is a tagcloud of the dominant keywords for a stable cluster from 2005 conference papers.

By examining the dominant keywords we manually recognize this cluster as related to aspect-oriented software development.

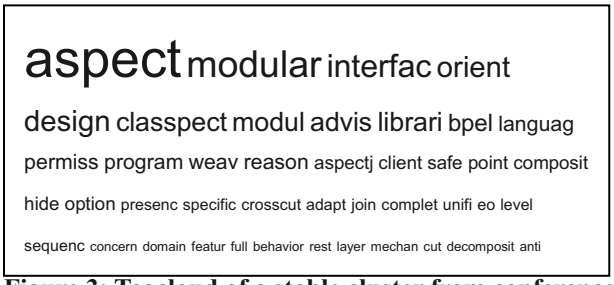


Figure 3: Tagcloud of a stable cluster from conference papers for 2005

Table 1 and Table 2 list some of the identifiable topics in each year's stable clusters for journal articles and conference papers, respectively.

Year	Distinct Cluster Topics
1996	HCI, lightweight source analysis, reliability
1997	real-time systems, OO, SCM
1998	inspection, protocol analysis,
1999	client-server, hypermedia, empirical, mobile systems
2000	exception handling, reliability, HCI
2001	embedded system, code decay, traceability, empirical
2002	codesign, attribute grammar, client-server, SCM, petri net, state machines
2003	agents, OO
2004	web services, exception handling, agile, OO metrics
2005	empirical, grammarware, automated reasoning, slicing
2006	Web, agile, requirements, OO metrics, empirical, middleware
2007	global software eng, metalock, ACSL, education, vulnerability, usability

Table 1: Identifiable topics for journal articles

Year	Distinct Cluster Topics
1999	aspects, agents, hypermedia, environments, smartcard, testing
2000	taxonomy, education, message sequence charts
2001	inspection, reliability, XML, product lines
2002	components, requirements, slicing, reliability
2003	OO, mixins, components, aspects, documentation
2004	architecture, MSR, FLAVERS[1], traits
2005	aspects, metrics, DOM, temporal logic
2006	spreadsheets, pointer analysis, distributed systems, SQL injection
2007	OPIUM[2], malware, agile, semantic query

Table 2: Identifiable topics for conference papers

[1] FLAVERS is a finite state verification tool and

[2] OPIUM is a package installer tool.

4.3 Label Classification Results

To validate the distinctiveness of the detected stable clusters, we checked to see if there are additional papers from other years that would have been classified within each of the stable clusters by training a labeled classification model using RapidMiner. Table 3 shows the results for the conference clusters.

	99	00	01	02	03	04	05	06	07
99	0	+9	+4	+3	+8	+6	+9	+8	+11
00	0	0	0	0	0	0	0	0	0
01	+1	+2	0	+2	+1	+2	+1	+2	0
02	0	0	+1	0	0	0	+1	0	0
03	0	0	0	0	0	0	0	0	0
04	0	0	0	0	0	0	0	0	0
05	0	0	0	0	+1	0	0	0	0
06	0	0	0	0	0	0	0	0	0
07	0	0	+1	0	+2	0	+1	+2	0

Table 3: Additional documents added to stable clusters from conference papers

The results indicate that, for most years, no additional papers could be found which closely match the papers in the stable clusters. This indicates that, not only are the document clusters distinct, their contents are very specialized. The one exceptional case in 1999 was due to the broad topic of testing which matched many papers in subsequent years.

A similar analysis was conducted on the stable clusters of journal articles. In this case, there were a lot more matches across different years, but mainly because of the presence of the empirical studies topic, which tends to crosscut many techniques.

4.4 Effect of cluster size on stability

While we are able to identify many topics, it is evident from examining the original set of papers that more topics are missed. One observation is that stable clusters also tend to be small. This is borne out by closer examination of cluster sizes.

Figure 4 shows, for each cluster of conference papers, its size versus the number of runs between when it was formed to when it was merged.

As we can observe from this figure, stable clusters (clusters with runs > 10) are also small in size, averaging less than 5 documents. This indicates that the agglomerative clustering employed has a tendency to pick up specialized topics. The data for journal articles follows a similar pattern.

The preceding results indicate that the stable clusters do not capture all the important topics for a given year. This implies the need for other classification algorithms to make additional identification.

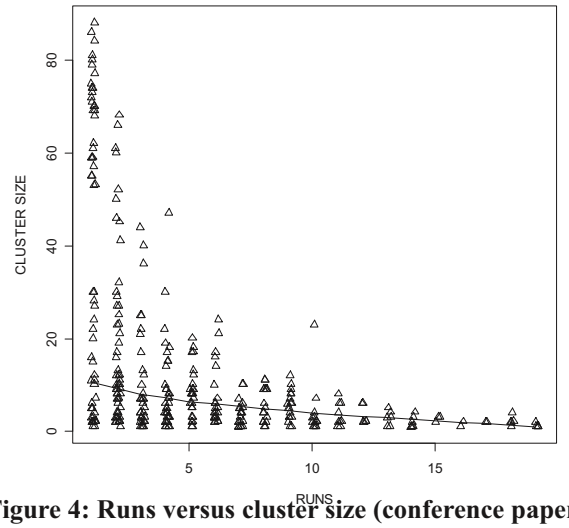


Figure 4: Runs versus cluster size (conference papers)

4.5 Relationships between conferences and journals

Based on the observation from Table 1 and Table 2, we do not see a pattern where conference paper topics serve as “advance notice” of journal papers. One of the reason should be that the stable clusters are mostly small sized clusters which cannot represent the main theme of certain year. And at the same time, it is possible that the conference papers would end up in more specialized journals other than the ones listed here.

5. Summary and Future Work

In this paper, we applied agglomerative hierarchical clustering over research papers from software engineering journals and conferences. We analyze the resulting clusters for meaningfulness by using a measure of stability. While the stable clusters found are indeed meaningful, they only identify a subset of the papers. Several other hierarchical clustering algorithms such as partitional and constrained agglomerative algorithms have been shown to provide improved results. [13] We plan to apply these other algorithms and determine the usefulness of our stability measure to find meaningful clusters from the results of these algorithms.

Work in this study is focused on empirical data for "meaningful" clustering in certain research area—software engineering, and interesting keywords analysis is executed from the perspective of software engineering experts but not data mining experts, so there is still a lot work to do on certain data mining perspectives.

- Evaluate quality of agglomerative hierarchical clustering based on distance to centroid of each data point, then calculate the quality of each level of hierarchical clustering based on the methods from [13].
- Expand the analysis to cover a larger range of papers from earlier years.
- Mining association rules [7] from software engineering literatures to predict research designs, activities and possible results.

And [3] provides us another dimension to explore, the co-authorship among active researchers, which is still an interesting topic. For example, it is generally assumed that certain researchers just work on several limited research areas and possibly interdisciplinary areas concerning those research areas, so the name of authors could be a potential source of information for searching certain research topics.

6. Acknowledgement

The authors here would like to express our gratitude and thanks to Professor Parvathi Chundi, Professor Zhengxin Chen and Nian Yan for their valuable assistance on text mining knowledge and techniques.

7. References

[1] N. O. Andrews, E. A. Fox. Recent developments in document clustering. Technical Report TR-07-35, Department of Computer Science, Virginia Tech, Oct. 2007. http://eprints.cs.vt.edu/archive/00001000, accessed Mar. 10, 2008.

[2] P. Ferragina, A. Gulli. The anatomy of snaket: A hierarchical clustering engine for web-page snippets. In Boulicaut, J.-F., Esposito, F., Giannotti, F., and Pedreschi, D., editors, PKDD, volume 3202 of Lecture Notes in Computer Science, pages 506–508. Springer.

[3] A. E. Hassan, R. C. Holt. The small world of software reverse engineering, Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04).

[4] G. Lee, X. Wu, J. Chon. Rearranging Data Objects for Efficient and Stable Clustering, *SAC'05*, USA, pp.519-523, March 13–17, 2005, Santa Fe, New Mexico.

[5] L. Massey. On the quality of ART1 text clustering, Neural Networks, Volume 16, Number 5, June 2003 , pp. 771-778(8).

[6] Q. Mei, C. Zhai. Discovering evolutionary theme patterns from text – an exploration of temporal text mining, KDD'05, August 21-24, 2005, Chicago, Illinois, USA.

[7] T. Nasukawa and T. Nagano。 Text Analysis and Knowledge Mining System，IBM Systems Journal 40, No. 4, 967–984 (2001).

[8] A. W. Rivadeneira, D. M. Gruen, M. J. Muller and D. R. Millen. Getting our head in the clouds: toward evaluation studies of tagclouds. Proc. of the SIGCHI conference on Human Factors in Computing Systems (CHI '07), 2007.

[9] M. Shewhart, M. Wasson. Monitoring a newsfeed for hot topics, KDD-99 San Diego CA USA.

[10] J. Stefanowski, D. Weiss. Comprehensible and accurate cluster labels in text clustering, RIAO'2007 Conference, Pittsburgh, PA, USA, 2007.

[11] M. Wurst. The word vector tool and the RapidMiner text plug-in. http://wvtool.sf.net, accessed July 31, 2007.

[12] O. Zamir, O. Etzioni. Grouper: a dynamic clustering interface to Web search results. Computer Networks (Amsterdam, Netherlands: 1999), 31(11–16):1361–1374.

[13] Y. Zhao, G. Karypis. Hierarchical clustering algorithms for document datasets, Data Mining and Knowledge Discovery, Vol. 10, pp.141-168, 2005.

A Model-Driven Approach for the Semi-Automated Generation of Web-based Applications from Requirements

Ali Fatolahi, Stéphane S. Somé, and Timothy C. Lethbridge

School of Information Technology and Engineering, University of Ottawa

This paper presents a semi-automated method for the generation of web-based applications from high-level requirements in accordance with model-driven architecture (MDA). MDA is a relatively new paradigm, which aims at providing a standard baseline for model-driven development. The ultimate goal of MDA is to (semi)automate the process of software development from requirements to code using an interoperable set of standards. In this paper, we present a method to generate web-based applications from requirements expressed as use cases using MDA-based tools, techniques and methods. The use case model is used as a baseline to generate other models including a state machine and a user interface model, which are eventually transformed into a platform-specific model used for code generation.

***Index Terms*— Requirements, MDA, Use Case, State Machine, User Interface, Mapping**

I. Introduction

MDA [1] is the OMG's [2] solution to increase model reusability and design time interoperability. A very important feature of MDA is a facility to transform models. MDA provides a collection of popular standards beneath a common philosophy to alleviate the process of quality software design and implementation.

According to [3], there has been a growing interest in MDA within the software community in the recent years; even the US government has held some workshops on employing MDA [4]. As another instance, IBM has developed its latest software engineering suite: Rational Software Architect [5] based on MDA. This is also true about many other tools such as MagicDraw [6], ArgoUML [7], Enterprise Architect [8] and System Architect [9].

The MDA process starts with capturing requirements at a computation-independent layer [3]. In our approach use case descriptions are used for requirements capture. Use cases are a popular technique for systems analysis and design. They are mainly used in textual form.

However, since writing textual descriptions is not as formal a task as drawing UML [10] models and writing programming code, various guidelines have been proposed to ease the process of writing use case descriptions and benefiting from these documents (e.g. [11] and [12]). The fact that several approaches have examined semiautomatic use case based tools and techniques (e.g. [13], [14] and [15]), evidences that employing use cases for analysis would be more useful if we found more systematic ways to connect them to lower level design models or even the code.

In this paper, we present a method for the semi-automated generation of web-based applications from requirements. Requirements are expressed as use case descriptions along with a domain model supporting the use cases. The whole model is used to produce a state machine. A default user interface model created based on the state machine is refined by the developer to form the desired user interface of the application. Based on these models, the method generates a platform-specific model which is used to generate the code. The 'user' of our method is the developer.

In order to perform the transformation, a set of mapping rules are defined. This set plays a critical role in the method as it guarantees the completeness of the information required for code generation.

In order to assess the feasibility of the approach we have implemented the method using UCEd [16] and AndroMDA [17]. UCEd is used for use case modeling and AndroMDA is employed for code generation. In addition, we have developed an application to generate the platform-specific model. However, the method and supporting tools and techniques are supposed to be extensible to higher level requirements and adaptable with other tools.

The rest of this paper is organized as follows. In Section 2, technical background terms of this research are explained. Section 3 addresses related research and practice in past and how they are related to this research. Section 4 carries the elaboration of our method and the applied tools and techniques along with a case study. In Section 5, we, briefly provide a conclusion; discuss some research issues and present our plan for future work.

II. Background

According to [18] a use case is a 'description of the sequence of behaviorally related transactions that user performs in order to have a dialogue with the system'. Use cases are expressed through use case models and use case descriptions. Use case description contains a natural language specification of the use case. In this research we continue working with the format used in [13]. According to this format, a use case description is composed of different sections including:

- **Title**
- **Precondition** that should be true before the use case starts
- **Steps** describing the behavior; each step could be described with an operation, an optional extension point

and a set of alternatives after the step

- **Use Case Alternatives**, describing the alternatives that are addressed within the steps
- **Post-condition**, which must hold after the use case completes.

A state machine is a 'behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events' [19]. State machines could be visualized using state/activity diagrams. As is asserted in [19], 'one may use state machines to model the behavior of any modeling element, most commonly, a class, a use case, or an entire system'.

A state machine is basically composed of states and transitions. A state is 'a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for events'. A transition is 'a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied' [19].

The state machines as we deal with, in this paper, are behavioral state machines, which are used to define the behavior of modeling elements, according to [20].

MDA is an effort by OMG, in order to standardize model driven software development [21]. It can be seen as a framework composed of four different layers of modeling. The topmost layer is the layer of Computation-Independent Models (CIM). CIM represents models that are valid in spite of the computational options. Then we have the layer of Platform-Independent Models (PIM). PIMs represent systems and software design and architecture; however, they do not contain any information about specific platforms. The third layer, Platform-specific Models (PSM) deal with the technological details of platforms. Here, logical design models are expressed in terms of certain platforms. At the lowest level, there are Implementation-Specific Models. These are real-world objects and components, acting as a running version of the system.

The Meta-Object Facility (MOF) [22] is the heart of MDA [23]. MOF provides a means of building new modeling languages or transforming different languages each to the other. The MOF is composed of very simple but strong-enough elements to describe any other modeling language. Although MOF does not provide any specific notation, it is possible (and convenient) to use basic UML Class modeling notations (with few considerations) to depict MOF models.

MDA admits two levels of MOF-based languages [24]. The first level addresses languages rooted in the MOF itself. The second level deals with the UML profiles. This level involves different UML extensions. In order to facilitate model exchange amongst different tools and standards, an XML Metadata Interchange [25] format has also been defined as a part of MDA.

MDA could also be understood by its meta-modeling mechanism, which is reflected in Table 1.

TABLE I
MDA META-MODELING MECHANISM

M3 (MOF)	Metametamodeling layer, including the most abstract materials required to build new languages and interoperability standards
M2 (UML, CWM, ...)	Metamodeling layer, providing the notation and formalism that can be used to model specific domains and systems. This layer is fed by M3. Examples are UML profiles.
M1 (User Model)	Projections of M2 in terms of certain user requirements. This includes di erent extensions of M2 to model the specifications of a certain subject.
M0 (Runtime Model)	Runtime objects. Running versions of M1.

III. Related Work

Nguyen and Chun [26] describe a method to make UML models more dynamic. They present a restricted use case specification language that is related to domain objects using some key words. They also use the notion of aspects [27] in order to describe how their method generates sequence diagrams. They build design diagrams that are connected to MDA-based metadata through hyperlinks and exchange languages. They also bridge UML sequence diagrams with current legacy code in order to raise the level of code reuse. Sequence diagrams are generated automatically but with the cost of making the use case specification language more complex by the inclusion of some design issues.

The main objective of Nguyen and Chun's work differs from ours in several ways. The focus of Nguyen and Chun's work is on building interactive UML design diagrams and code reuse. Our method is, however, focused on the whole MDA-based process from requirements to code. Nguyen and Chun use sequence diagrams as the core model, mainly because sequence diagrams could be easily connected to code for further reuse. We use state machines instead, which could be used in different levels of abstraction. Finally, In Nguyen and Chun's work, MDA standards are not used for interoperability purposes amongst different tools but as an internal way to model storage and exchange.

The closest work to ours is that of Wu *et al* [28]. Wu's work describes a method to generate a user interface code following MDA transformation and the Model View Controller (MVC) pattern. The method spans the gap from requirements to code for a user interface model by transforming boundary objects resulting from a robustness analysis [29] to JSP pages [30]. In order to do this, Wu et al provide a framework that starts with use case modeling and activity diagrams. Then they perform a robustness analysis to categorize the participating objects. Finally, JSP pages are built according to the transformation rules and UML models.

Unlike the work of Wu et al [28], our study covers the generation of code for the whole software system, not only the user interface part. Although, we select certain platforms to implement our method, the method itself is theoretically platform-independent. Finally, we have developed not only a method but also a practice to show the feasibility of the method, which is not present in [28].

We may also mention the work of Pastor and Molina [37], which is a conceptual framework for MDA-based software development environments. The approach presented in their book is neither rendered in practice nor formal, however since an ultimate goal of our method could be the generation of an MDA-based environment, Pastor and Molina's work provides a useful point of reference.

In our research, we use the method presented in [13], which is performed with the help of the UCEd. This method elaborates the necessity to support use-case-based requirements engineering. This support is given throughout domain objects, operation pre- and post-conditions, and semi-natural language use case steps. For each of the latter, UCEd provides some automatic and semiautomatic means. The output is a state machine that belongs to the category of platform-independent models, since it sketches an overview of how the system works without any design-related details.

UCEd provides a set of tools for defining use case descriptions, inclusion and extension. It also provides a semi-automated wizard for domain extraction. The resultant domain model may be refined by editing the information related to domain objects and their operations and attributes. UCEd can generate the corresponding state machine and simulate its execution.

In order to work with UCEd, one needs to first enter use case descriptions. Having this description validated, one proceeds through a wizard in which UCEd provides one with a series of different choices for domain objects. The result is a validated domain model. This domain may be optionally supplied with operations' conditions that are used to build operation contracts [33]. State machine can be generated now.

We also use AndroMDA. AndroMDA is an MDA-based code-generation framework. It provides a set of profiles for different platforms along with some mapping functions to transform models to each other or to the programming code. The process starts with user creating a starter application; this includes identifying the platforms to work with (e.g. J2EE, Struts, etc.). As a result an empty UML model is built, which includes required references to UML profiles of technology and platforms. The developer then imports this model into a UML tool and adds design elements needed by the AndroMDA framework. The resulting model is processed by AndroMDA leading to executable code, which may be refined by the developer.

IV. The Method

The solution we provide in here is a method, which is both model-driven and requirements-based. The input is provided through use cases and the output is the executable code generated in accordance with MDA. Different steps of the method are either automatic or semi-automatic. The whole process is actually a collection of mappings in accordance with XMI format necessities, MOF-based meta-models and MDA transformation rules.

As Figure 1 shows, the main task of this process is to transform a PIM to a PSM. This process is done through three main steps. First, a default UI model is created according to the state machine found in the PIM. The developer is then asked to refine this model to build the desired UI model. Finally the UI model, along with other parts of the PIM, is used to generate a PSM.

A working example taken from [17] is used in this section to elaborate the method using more details. The whole application is called 'Time Tracker' but we only cover one use case, named 'Search Timecards'.

A. First Step: CIM to PIM

Use case descriptions and default domain objects are considered as the CIM of our method. The objective of this step is to transform the CIM to the PIM. The PIM includes the state machine, the user interface model and the refined domain model.

Table 2 includes the information regarding the CIM for 'Time Tracker'. This CIM is transformed to the PIM

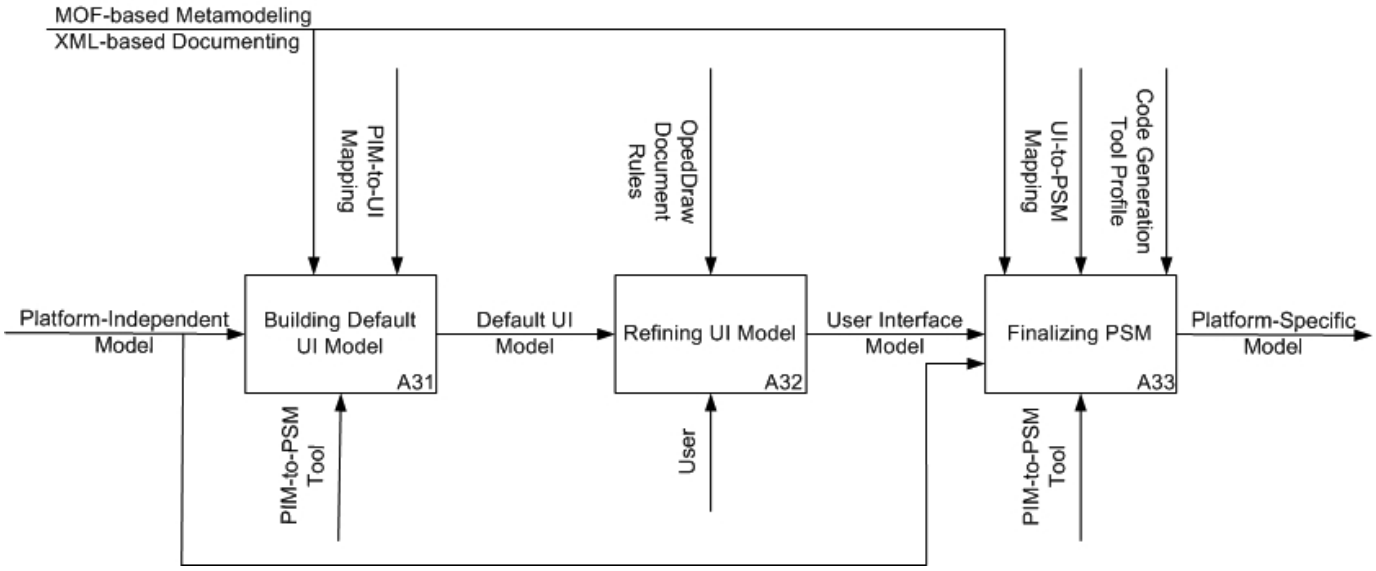


Fig. 1. Transforming a PIM to a PSM

TABLE 2
CIM OF TIMETRACKER

Use Case Name	Search Timecards
Use Case Precondition	Timetracker is Up
Steps	1. User browses Search Timecards page 2. Tmetracker populates Search Timecards page 3. User enters search parameters 4. User presses search button 5. Goto step 1
PostCondition	Search Timecards page is browsed

afterwards. Figure 2, shows a part of the PIM, which is a state machine. As this figure shows, transitions are named after the operations and their conditions. Other parts of the domain models are not shown for the sake of simplicity.

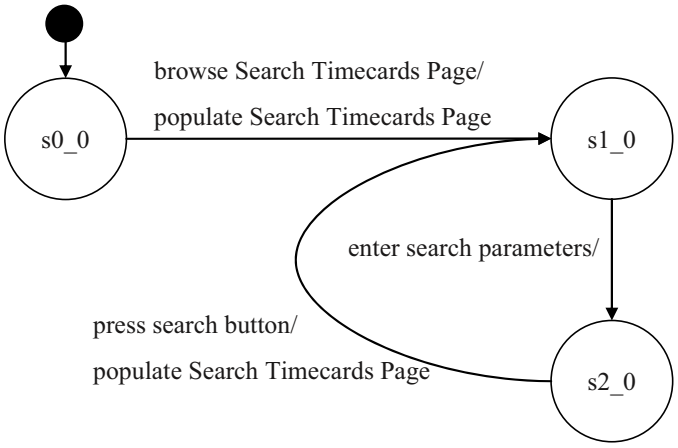


Fig. 2. The state machine generated by UCEd for the use case Search Timecards

The role of this state machine is critical, since it provides the mechanism to integrate different use cases. This is especially true about use cases that are related to each other by inclusion or extension mechanisms and use cases that act as a pre-condition to other ones. The state machine is also used as the origin of the default UI model described in next section.

UCEd provides two different ways of creating a state machine. The simpler is to generate the state machine based on the use case flow. However, the use case flow does not always reflect the exact flow of events. It is also possible to define operation conditions and the effect they have on domain objects in terms of changing their state. UCEd can use this information to generate another type of state machine based on the operation effects.

B. Second Step: PIM to PSM

A PIM-to-PSM is a mapping from a platform-independent model of domain objects, state machines and user interface models to a platform-specific model. The PIM generated in last step provides the information regarding the domain objects and their operations, the flow of events and the conditions that apply to every phase.

The true mapping should be done according to a selected profile. However, due to implementation constraints, we have to pick up some profiles that are already created to be used with a specific code generation framework, AndroMDA. The chosen profile contains various definitions about Java as the programming language, AndroMDA as the code generation framework, ArgoUML as the modeling tool, Struts [31] as the user interface framework and MySQL [32] as the database server.

Prior to the description of the method, it is necessary to define some terms regarding this platform:

- A controller class is the class responsible for controlling general activities within a use case. A controller can act as an entry point to dispatch messages and operation requests to the right target. This is in accordance with a design pattern with the same name. For more information regarding the controller pattern, see [33].
- An action event is an event to be called when submitting a form from within a web page. Action events usually include parameters which are the input fields on the forms.
- A deferrable event is an event calling a controller operation. Deferrable events are used to assign states with operations.
- A page parameter is any output that is either shown or used for other output fields on the web page.
- A value object is an object to carry the required information between domain objects and the presentation or data access layer.
- FrontEndView is a state stereotype implying that the stereotyped state represents a web page.

For more information, one may refer to [34]

The approach is not fully automatic; we need to interact with the developer to see what is expected as a value for user parameters. In order to do this, we create a default UI model according to the provided PIM state machine. This default UI model will be the base model to generate the PSM thereafter.

The method is adjusted to work with the OpenDocument format [35]. OpenDocument is an XML-based format to define documents containing text or graphics. This means that the user interface model should be created in accordance with this format. Currently, we use OpenDraw [36] to draw the UI model. The developer is required to create the UI model following some rules:

- Every nonempty slide is considered a presentation state (e.g. a web page)
- Drawing items could be grouped to represent a group of related outputs on a page or more importantly an input action and its related items (e.g. a submit button)
- A frame is a symbol of an action
- A triangle represents a dropdown input
- A rectangle represents a plane text input
- A cloud could be grouped with any input item to identify its data type
- A ring can be grouped with an action denoting the called operation associated with the action
- A cylinder could be grouped with any input item declaring its data source containing the name of table and/or column

- If a group of outputs was combined with a cross, this would mean a table view output

States are recognized as either presentation (front-end) or logic states. For each presentation state, s_i:

- s_{i-1} is the state preceding s_i
- s_{i+1} is the state following s_i
- $t_{i-1,i}$ is the transition from s_{i-1} to s_i
- $t_{i,i-1}$ is the transition from s_i to s_{i-1}
- $t_{i,i+1}$ is the transition from s_i to s_{i+1}
- $t_{i+1,i}$ is the transition from s_{i+1} to s_i

Neither s_{i-1} nor s_{i+1} could be a presentation state. Currently we accept no multiple transitions out of a state, which means there would be just one form per web page.

In the 'Time Tracker' example, state s1_0 is a presentation state that is prototyped using the model of Figure 3. The default user interface model includes four empty slides per

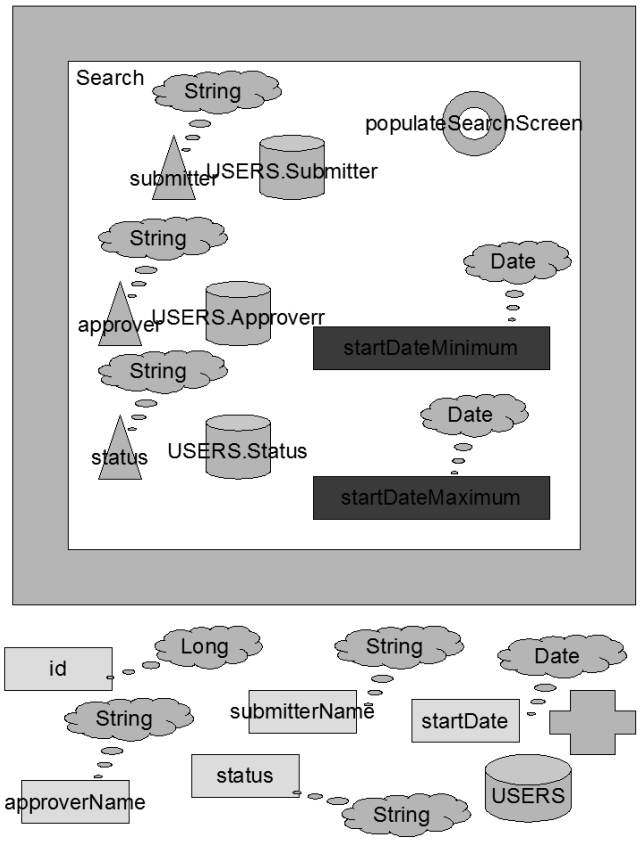


Fig. 3. The user interface model for Timetracker drawn by OpenDraw

four steps of the state machine in Figure 2. The rest of the UI model has to be defined by the developer. The developer has then generated the user interface model for this state machine assigning the model in Figure 3 to the step s1_0. According to this figure, Timetracker has a webpage containing two parts. First section is an action form submitting following values to the operation, populateSearchScreen:

- *submitter* from a dropdown input supplied by the database table USERS
- *approver* from a dropdown input supplied by the database table USERS
- *status* from a dropdown input supplied by the database table USERS
- *startDateMinimum* from a date input supplied by the User
- *startDateMaximum* from a date input supplied by the User

The second section shows the output that is a table containing the search results. Table columns are *id, submittername, approvername, status* and *startDate* all coming from the database table USERS.

Suppose A as the set of actions submitted from s_i. Each action is defined as the tuple $\{a, F_i, operation\}$. F_i is the set of input fields and is defined as $\{\{F,r\}|f,r\} +\}$ in which

- F is a set of fields
- f is a single field
- r is the value object providing the input
- *operation* is the name of the operation this action calls.

Also, assume O as the set of outputs shown by s_i. Each output is simply a set of output fields, F_o.

Back to the 'Time Tracker' sample, we can see that for s1_0, A includes one action that is defined as *{Search, {{submitter, USERS}, {approver, USERS}, {status, USERS}, {startDateMinimum, null}, {startDateMaximum, null}}, populateSeacrhScreen}*, where

- 'Search' is the action
- The set of input fields (F_i), includes
  - submitter, approver, and status coming from database table USERS
  - startDateMinimum, startDateMaximum entered by the User
- And populateSearchScreen, the controller operation to be called

There are some mapping rules that apply regardless of the UI model:

- There must be one controller class per use case
- There must be a service class per operation
- Controller classes must be dependent on their objects' service classes

In order to transform this UI model to a PSM model, we abide by some further rules.

- Every presentation state becomes a state stereotyped as *FrontEndView*
- For each member of A
  - a becomes an action event on $t_{i,i-1}$ or $t_{i,i+1}$ whichever exists
- a becomes a deferrable event on s_{i+1} calling operation
- *operation* becomes an operation of the controller class
- F_i becomes the set of input parameters on both a and *operation*
- For every database table referred to by members of F_i, an entity domain object and a value object are created
  - There would be a dependency from every entity domain object to the relevant value object
  - Add an operation to the service class to retrieve the data from database
  - Make a dependency from the service class

- For each member of O, F_o becomes the set of parameters on a signal event belonging to $t_{i-1,i}$ or $t_{i+1,i}$ whichever exists

In case of 'Time Tracker', the generated PSM will include a state machine containing three states. State s1_0 of Figure 3 will be stereotyped as FrontEndView. The transition out of this state has an action event called, *search* with the user parameters, *submitter, approver, status, startDateMinimum,* and *startDateMaximum*. The state preceding s1_0 calls a deferrable event named, *populateSearchScreen(submitter, approver, status, startDateMinimum, startDateMaximum)*. The transition to the state, s1_0 carries a tabular page variable of (id, submitterName, approverName, status, startDate).

C. Third Step: PSM to Code

This last step is done by the code generation tool. Currently, we generate a PSM that could be edited by ArgoUML and read by AndroMDA.

V. CONCLUSION

This paper reflects the present state of research and development of our method, so our main purpose has been to present the concepts of the current tool and method. Our method is a semi-automated approach for the generation of web-based applications from requirements. This is done using several transformations over use cases, user interface models, state machines, design models and code.

A java application has been implemented to run the method. This tool has so far been tried on several examples taken from actual case studies including the ones found in [17].

As future work, we intend to cover more complicated cases. We are especially interested in mixed problem classes, applications with database transactions and multiple use cases and state machines within the same application. We also intend to evaluate the effectiveness of the method by having group of developers evaluate it in practice.

We plan to improve the method itself by including tool and platform profiles. These profiles would guarantee that the method remains tool and platform independent. Profiles will be provided as a component of a general family of platform-independent mapping patterns.

REFERENCES

[1] ModelDriven Architecture, www.omg.org/mda, [Accessed 25 February 2008].

[2] Object Management Group, www.omg.org, [Accessed 25 February 2008].

[3] T. O. Meservy, K. D. Fenstermacher, "Transforming software development: an MDA road map." IEEE Computer, Vol. 38 (9), pp. 52–58 (2005)

[4] OMG, "Proceedings of the MDA in the US Government Workshop, Washington DC", 15 November 2005 [online], Available: (www.omg.org/docs/gov/051101.pdf). [Accessed 28 Feb. 2008]

[5] "IBM Rational Software Architect", www-306.ibm.com/software/awdtools/architect/swarchitect, 25 February 2008.

[6] "MagicDraw", www.magicdraw.com, [Accessed 28 Feb. 2008]

[7] "ArgoUML", argouml.tigris.org, [Accessed 28 Feb. 2008]

[8] "Enterprise Architect", www.sparxsystems.com.au/products/ea.html, [Accessed 28 Feb. 2008]

[9] "Telelogic System Architect", www.telelogic.com/Products/systemarchitect, [Accessed 28 Feb. 2008]

[10] "Object Management Group UML", www.uml.org, [Accessed 28 Feb. 2008]

[11] A. Cockburn, *Writing Effective Use Cases*. AddisonWesley, Harlow, 2001.

[12] Y. Chen, I. Y. Song, "Guidelines for Developing Quality Use Case Descriptions." Proceedings of 2007 IRMA International Conference, pp 564–567 (May 2007)

[13] S. S. Som´e, "Supporting use case based requirements engineering" Information and Software Technology, Vol. 48 (1), pp. 43–58 (2006)

[14] H. Behrens, "Requirements Analysis Using Statecharts and Generated Scenarios" In Doctoral Symposium at IEEE Joint Conference on Requirements Engineering (2002)

[15] A. G. Sutcliffe, N. Maiden, A. M. Minocha, D. S. Manuel "Supporting ScenarioBased Requirements Engineering" IEEE Transactions on Software Engineering, Vol. 24 (12), pp. 1072–1088 (1998)

[16] "Use Case Editor", www.site.uottawa.ca/˜ssome/Use Case Editor UCEd.html, [Accessed 25 February 2008]

[17] "AndroMDA", www.andromda.org, [Accessed 25 February 2008]

[18] I. Jacobson, *Objectoriented software engineering : a use case driven approach*, AddisonWesley, 1993.

[19] G. Booch, R. Rumbaugh, I. Jacobson, *The unified modeling language user guide*. AddisonWesley, Upper Saddle River, 2005.

[20] "UML, Unified Modeling Language: Superstructure version 2.1.1", 2007 02 05, [online]. (Available: www.omg.org/cgibin/doc?formal/07-0205), [Accessed 25 Feb. 2008]

[21] T. Stahl, M, Volter, J. Bettin, A. Haase, and S. Helsen, *Modeldriven software development : technology, engineering, management* /translated by Bettina von Stockfleth. John Wiley, Chichester, England ; Hoboken, NJ. 2006.

[22] "Meta Object Facility (MOF) Core Specification OMG", [online]. (Available: www.omg.org/technology/documents/modeling spec catalog.htm#MOF). [Accessed 25 Feb 2008]

[23] "OMG's MetaObject Facility (MOF) Home Page", www.omg.org/mof, [Accessed 25 Feb 2008]

[24] D. S. Frankel, P. Harmon, J. Mukerji, J. Odell, M. Owen, P. Rivitt, M. Rosen, R. M. Soley, "The Zachman Framework and the OMG's Model Driven Architecture, a BP Trends Whitepaper", September 2003. [online] (Available: www.omg.org/mda/mda files/0903WP Mapping MDA to Zachman Framework1.pdf), [Accessed 25 Feb 2008]

[25] "XML Metadata Interchange", www.omg.org/technology/documents/formal/xmi.htm, [Accessed 25 Feb 2008]

[26] P. Nguyen, and R. Chun, "Model driven development with interactive use cases and UML models". Proceedings of The International Conference on Software Engineering Research & Practice and Conference on Programming Languages & Compilers, pp 534–540 (2006)

[27] I. Jacobson, Ng. PanWei, *Aspect oriented software development with use cases*. AddisonWesley, Upper Saddle River, NJ (2005)

[28] J. H. Wu, S. S. Shin, J. L. Chien, W. S. Chao, and M. C.L Hsieh, "An Extended MDA Method for User Interface Modeling and Transformation". Proceedings of The 15th European Conference on Information Systems. pp 1632–1641 (2007)

[29] D. Rosenberg, and M. Stephens, *Use case driven object modeling with UML : theory and practice*, Apress Publishers (2007)

[30] "Java Server Pages Technology", java.sun.com/products/jsp, [Accessed 25 Feb 2008]

[31] "Struts", struts.apache.org, [Accessed 25 Feb 2008]

[32] "MySQL AB", dev.mysql.com, [Accessed 25 Feb 2008]

[33] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall PTR (2005)

[34] "AndroMDA BPM4Struts", galaxy.andromda.org/docs/andromda-bpm4strutscartridge/index.html, [Accessed 25 Feb 2008]

[35] "OASIS", www.oasisopen.org, [Accessed 25 Feb 2008]

[36] "Draw", /www.openoffice.org/product/draw.html, [Accessed 25 Feb 2008]

[37] O. Pastor and J. C. Molina, *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*, Springer (2007)

A model-driven toolset to support an approach for analyzing integration of business process aspect of enterprise application integration

Souvik Barat
Tata Research Development & Design Centre,
Pune, India
souvik.barat @tcs.com

Vinay Kulkarni
Tata Research Development & Design Centre,
Pune, India
vinay.vkulkarni@tcs.com

Abstract

The demand for integrating enterprise applications is growing as a consequence of the need to support dynamic cross-functional inter-organizational business processes. Semantic underpinnings of the high level notations used by Industry practice to specify enterprise applications, and tools and techniques prevalent in industry are not rich enough so as to be able to capture and verify the required properties of interest with rigor. As a result, ensuring semantic correctness of integration of existing applications is a manual effort intensive process. By combining the usability of high level notations prevalent in industry practice and the analytical rigor of formal techniques, and visualizing the integration problem as a view-integration problem over data, service and process models, the problem can be addressed more pragmatically. We present such an approach for view-integration of process models wherein the principal objective is to analyze the process views in the context of reusability, adaptation and integration, and a model-driven toolset to automate the approach.

Keywords: Business Process Integration, Model-driven Integration, Model-driven Framework.

1. Introduction

Enterprises are witnessing an increased thrust on collaboration and integration of existing applications to fulfill the upcoming business demands. In this integrated environment, the enterprises are no longer limited to a specific organization or department but span across the entire value chain to provide value added services [14, 16, 25]. Typically, enterprises are organized into a set of departments each catering to a cohesive functional need with IT systems providing automation support to the extent possible. As a result, over a period of time, an enterprise ends up with a set of isolated applications providing point solutions each constructed for a specific purpose. Integration of such disparate applications to realize the desired requirements with maximal reuse is a critical challenge for modern enterprises.

We model enterprise application as a 3-tuple comprising of its data, service and process models [15]. The application integration problem can now be visualized as a view-integration problem over data, service and process models. Present Enterprise Application Integration (EAI) [16] solutions only provide 'plumbing support' for data adaptation and for correct invocation of services in the light of the integrated data models. Very little work of practical significance is seen with regard to process level integration [14]. Industry practice uses a set of modeling notations such as BPEL [13], UML profile for business process [12] etc. to specify business processes. Semantic underpinnings of these notations not being rich enough, they cannot be used for verifying correctness of process integration. As a result, industry practice has to depend only on testing – an effort and time intensive activity. On the other hand, a variety of formal techniques [3, 8, 9, 19, 26] using a variety of formalisms [1, 17, 10] have been proposed to address process integration problem. However, these approaches have not seen wide industrial acceptance as practitioners find them too involved to use and too detailed to specify as compared to the high level notations they are used to. Therefore, there is a need for a pragmatic approach that combines the rigor due to formal techniques and the usability and high level of abstraction due to prevalent industry practice to address the integration problem in a comprehensive manner.

This paper proposes a pragmatic approach for analyzing business processes for integration properties of interest, and a model-driven framework for automating the approach. We present, process automata [4] as an abstraction to formally represent a process view, a set of integration properties of interest, and a set of mediation operators to be used for adaptation of a process view in conflict for a class of mismatches. The rest of this paper is organized as follows: Section 2 presents the approach. Section 3 provides an overview of the formal foundations of the approach. Overview of a model-driven framework is presented in Section 4 and section 5 concludes outlining future work.

2. Approach

Enterprise application integration is about realizing the desired integrated application through maximal reuse of existing applications. Consider $A_{desired}$ is the desired application that needs to be realized through maximal reuse of existing applications A_1, A_2,…A_n. For safe integration, it needs to be established whether an existing application A_i fits into the context of $A_{desired}$ or not. In case of a mismatch, one would like to know if the existing application could be made to fit with some adaptation. The fitting applications, with or without adaptation, can be considered for integration in order to realize the desired application with assurances of completeness of the integration.

We visualize an enterprise application being specified in terms of its *data*, *service* and *process* models that we term as views. Since an application is designed to operate in a specific, and typically, isolated context, built-in assumptions may sneak in the definition of these views. We argue that these assumptions lead to conflicts or mismatches while integrating these applications. The fundamental issue in enterprise application integration lies in identification and mitigation of these conflicts. The conflict or mismatches can be identified by verifying a set of properties of each of these views. We term these properties as *integration properties*. The identified conflict/mismatch can be mitigated by transforming the view in conflict. We term this transformation as *mediation*. Since modeling notations used by industry practice to specify these views are not amenable for rigorous analysis, we introduce a set of formal models for each of these views to enable formal analyses. We term these formal models as *analysis specific models*. Figure 1 depicts this separation of concerns. We use model-2-model transformation specifications [24] as a means to provide bi-directional transformation between

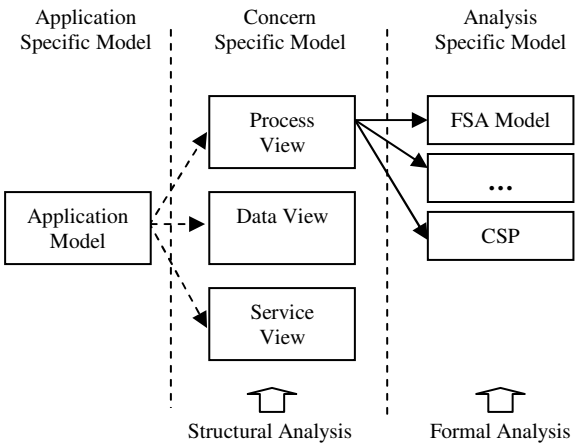


Fig 1. Model decomposition & transformation

these models. Decomposition of an application into views and transforming these views into a set of analyzable models provide an opportunity to use specialized tools and techniques in the respective domains. We consider only the behavioral aspect of the enterprise application in this paper.

The behavioral aspect of an enterprise application is typically a control flow over a set of process steps. A process step could either be a manual task or automated through a service offered by an application. We propose two integration properties, *compatibility property* and *completeness property*, for process view integration. *Compatibility property* verifies whether a process view of an existing application A_i fits into the context of the process view of the desired application $A_{desired}$ or not. Essentially, this property analyzes the control flows over a set of process activities of two process views in the context of reusability. The *completeness property* ascertains the behavioral completeness of a set of fitting process views of existing applications with respect to the process view of the desired application. This ensures the required process activities of the desired process view are present in any of the participating process views and they do not violate any flow assumptions, and also determines the gaps.

To enable rigorous analysis, we use process automata as a formal representation of a process view and use a set of mediation operators to adapt a process view in conflict. The process automaton is based on finite state automata [11] model. The concept of simulation relation [17] and language containment [11] are adopted appropriately with the notion of refinement [2] for ascertaining the proposed integration properties. The mediation operators are based on the concept of refinement, abstraction and hiding. We use a popular

formal toolset [7, 18] for automating the analyses. We have developed a model-driven framework that combines the rigor due to formal techniques and the usability and high level of abstractions due to prevalent industry practice to address the process integration problem in a pragmatic manner.

3. Formal foundation

Fundamentally, the identified integration properties are kinds of checking refinement [2] using system equivalence [17] and their pre-order relations. The relations can be verified if the involved systems can be visualized as a label transition system. By modeling the behavioral aspect of enterprise application, process view, as deterministic finite state automata, termed as process automata [4], the verification of integration properties can be visualized as the automata synthesis problem. Due to the lack of space, we describe only the important definitions and concepts in this paper. The detail formal interpretation can be found in [4] and [5]

3.1 Process automata

Formally, we represent a process view as a deterministic finite state automaton wherein process states are the states of automaton, process activities are the alphabets or the events, and activity flows are the state transition relations. Parallelism and synchronization between activities is addressed by flattening out the possible interleaving of activities. The conditional expressions are also considered as the events and transitions of the transition system. A process automaton P is a 5-tuple: P (S, E, T, s, F) where

S is a finite non empty set of process states.

E is a finite set of events, which represent the vocabulary of process activities and conditional expressions. The process activities and conditional expressions are suitably mapped onto the events, i.e. services, manual task and conditional expression over process data can be visualizes as event of a process automata.

T is a non empty set of transitions. A transition $t \in T$ is defined as, $s \xrightarrow{e} s'$, where s is the source state, s' is the target state and e is an event. The structural constructs between process activities and conditional expressions are mapped into the transitions relations.

s is the start state ($s \in S$).

F is the set of final states ($F \in S$).

We assume there is no incoming transition to the start state and final states have no outgoing transitions.

3.2 Analysis technique

In this section, we present the important definitions to establish proposed integration properties.

Mapping (M: $E_1 \rightarrow E_2$)

In a real life integration scenario, different applications may use different terminologies that reflect in different vocabularies being used to describe their process views. Mapping M describes a correspondence between a set of events E_1 of process P_1 to a set of events E_2 of process P_2.

$e_1:P_1 \rightarrow e_2:P_2 \Rightarrow$ Event e_1 of process P_1 and event e_2 of process P_2 are semantically equivalent events. We define a mapping function (f_{map} (event)), which returns either the corresponding event e_2 of E_2 for a given event e_1 of E_1 from the provided M if event exists in M or the event itself.

Restriction (P1, P2, M)

Given a mapping M, the *Restriction* of a process P_1 by a process P_2 results in a process P_R that contains only those transitions of P_1 whose corresponding events are present in process P_2, i.e. P_R = Restriction (P_1, P_2,M), where

Restriction (P_1, P_2, M) = Ignore (P_1, ($E_1 - M(E_2)$))

Restriction operator ignores the transitions of P_1 having events from the set ($E_1 - M(E_2)$). Given a process P and a set of events I, Ignore operator computes the transitive closure graph by considering the set of transitions triggered by $e \in I$ as epsilon moves and constructs an equivalent deterministic finite automaton using subset construction algorithm.

Composition setting

Formally, a composition setting can be described as a 3-tuples (P_D, P_C, M), where $\mathbf{P_D}$ is process automaton of the process view of desired application $A_{desired}$, $\mathbf{P_C}$ is a set of process automata {P_1, P_2…P_n} of process views of all participating applications A_1, A_2,…A_n, and M is set mappings {M_1, M_2, …,M_n} respectively, where M_i denotes the mapping between participating process automata and desired process automata

Event Completeness

The event completeness criterion holds for a composition setting CS = (P_D, P_C, M) iff it satisfies the following condition

$$\forall e \in E_D, \exists e' \in E_C \text{ such that } (e = f_{map}(e'))$$

Simulation relation

Given a mapping M, two processes $P_1 = (S_1, E_1, T_1, s_1^0, F_1)$ and $P_2 = (S_2, E_2, T_2, s_2^0, F_2)$, a relation $R \in S_1 \times S_2$

is called a simulation relation iff it satisfies the following condition:

$$\forall s_1 \in S_1,\ s_1' \in S_1 \text{ and } s_2 \in S_2$$
$$\text{if } (s_1, s_2) \in R,\ s_1 \xrightarrow{e} s_1' \text{ then}$$
$$\exists s_2' \in S_2 \text{ s.t. } s_2 \xrightarrow{f_{map}(e)} s_2' \wedge (s_1', s_2') \in R$$

A process P_2 simulates process P_1, denoted by $P_1 \leq_M P_2$, if there exists a simulation relation $R \in S_1 \times S_2$ such that $(s_1^0, s_2^0) \in R$.

Collective simulation relation

Let (P_D, P_C, M) is a composition setting where P_D = (S_D, E_D, T_D, s_D^0, F_D) is desired process automaton, P_C = (S_C, E_C, T_C, s_C^0, F_C) is collective process automaton of process automata $P_1, P_2, \ldots, P_k$ and M is set of event mapping. A relation $R \in S_D \times S_C$ is called a collective simulation iff it obeys the following conditions:

$$\forall s_d \in S_D, s_d' \in S_D \text{ and } s = \{s_1, s_2, ..., s_k\} \in S_C$$
$$\text{if } (s_d, s) \in R,\ s_d \xrightarrow{e} s_d' \text{ then } \exists s' \in S_C$$
$$\text{such that } s \xrightarrow{f_{map}(e)} s' \wedge (s_d', s') \in R$$

Where transition over the set of state of participating process automata is as follows

$$s \xrightarrow{f_{map}(e)} s', \text{ there exists at least a i}$$
$$\text{s.t., } s_i \xrightarrow{f_{map}(e)} s_i', \text{ where } i \in [1, k]$$

A given a set of mapping M, product of participating processes P_C simulates the desired process P_D, denoted by $P_D \leq_M^C P_C$, if there exists a simulation relation R $\in S_D \times S_C$ such that there exists an $s^0 \in s_c^0$ that satisfies $(s_D^0, s^0) \in R$.

Compatibility property

A compatibility property is proposed to determine the extent of reusability of a process in the context of the desired process and the adaptation required in case of a mismatch. A process automaton P_1 is compatible with respect to process automaton P_2 iff following condition is satisfied:

Restriction (P_2, P_1,M)$\leq_M P_1$

Completeness Property

Completeness property ascertains that the desired process can be realized by integrating the participating processes. A given composition setting (P_D, P_C, M) is complete iff the following conditions are satisfied:

- Composition setting is safe i.e.
  $\forall i \in [1,k]$, where k = | P_C |, the compatibility criteria holds for process P_i with respect to desired process P_D i.e. Restriction(P_D,P_i,M_i)$\leq_M P_i$
- Composition setting satisfies the event completeness relation.

3.3. Process mediation operators

The proposed formal foundation provides a set of techniques for analyzing the compatibility property and completeness property. It also generates a counter-example in case of mismatches in execution orders of the involved activities between the existing and the desired processes. A certain class of conflicts can be resolved by transforming the process model in conflict into the desired process model using abstraction, refinement, hiding technique. For instance, a process activity of a specific process model can be realized as a sequence of process activities in desired process model (refinement), a sequence of process activities of a process model can be considered as a single process activity in desired process (abstraction), or an unintended process activity (like notification of a confirmation messages) is present in a process model, which is not required in desired process. We term such mismatch as resolvable mismatch. The other class of mismatch can not be resolved by transformation techniques, we term this class of mismatch as irresolvable mismatch. For instance, if the process activities violate the execution order, i.e. process activity *a2* is executing after the execution of process activity *a1* in a process model, and desired process model expects *a2* should be executed prior to the execution of *a1*.

3.4. Toolset

In order to automate the proposed formal techniques, we have implemented a toolset, *process automata verification environment (PAVE)*, using Esterel [6], fc2tools [7] and MONA package [18]. The toolset implements proposed operators and the relations by adopting various existing algorithms and formal techniques. The implemented toolset uses an extended FC2 [7] format as the input specification language.

4. Model-driven framework

We have automated the proposed approach through a model-driven framework that supports i) creation of the different kinds of models and bi-directional transformations between them, ii) specification of integration properties using a domain-specific language, and iii) analysis toolkit. Among the various models supported, *Application model* is high-level specification of the entire application, *Concern-specific*

model captures a specific concern of interest of an application model, and *Analysis-specific model* facilitates use formal verification techniques. All these models are MOF-describable and hence can be transformed from one to the other [24]. Analysis specific model contains two parts: structural part and behavioral part. The structural part conforms to a specific metamodel with a place-holder for behavioral part which is a textual specification conforming to a specific formalism. For want of space, we do not discuss details here which can be found in [5]. The core functional units of the proposed framework are as follows:

MDA Modeler: Provides support for defining various MOF-describable meta models and models. We use process view metamodel and process automata specific metamodel proposed in [5]. Process view metamodel is a concern-specific metamodel that is aligned with BPEL constructs for specifying business processes. Process automata metamodel is an analysis-specific metamodel whose behavioral part conforms to the input language of process automata verification language, i.e. FC2 format.

Property Specification Language: A domain specific language is proposed for specifying *integration properties* at a higher-level of abstraction [5]. These specifications are independent of the specific formalism being used for analysis. The expression language of the proposed language is based on Object Constraint Language (OCL) [23] and property specification constructs are based on first-order logic and predicate logic.

Model Transformation Environment: This unit deals with the basic requirement of the model transformation mechanism (conforming to QVT transformation). This provides an environment to specify the transformation rules, at the meta model level, to automate the transformation process. In order to establish the interoperability between the supported models, we define a set of transformation rules. The rules are:

1. Transformation between application model and process view model (Application2ProcessView): the process view metamodel is aligned with BPEL constructs. The transformation from UML Profile for business process to BPEL is defined in [12]. We use the same strategy for this transformation.
2. Transformation between process view model and process automata specific model: We define a bidirectional mapping strategy, *Process2Analysis transformation*, between process view metamodel and structural part of the process automata specific metamodel. The proposed schema is conforming to QVT model-to-model transformation strategy [24]. The translation of process view model to process automata specification (*Process2Fc2*) is a two steps process. First we translate the process view into Esterel program [6] using QVT model-to-text transformation language (SpeclL) [21], and then we use Esterel compiler and FC2Tools [7] to compile the Esterel program into FC2 format.

Model Based Verification Engine (MBVE): This is an environment for analyzing the MOF-describable model elements. Typically, it verifies the invariants over a set of model elements with or without transformation. We use OCL and their corresponding toolset to realize model based verification environment.
Analysis Engine: Analysis Engine is set of formal tools. We use PAVE as the formal toolset for reasoning proposed integration properties.

Analysis process

UML Profile for business process [12], conforming to BPEL constructs, is a visual language for specifying business processes. We use this as a modeling language for *application model* in our framework. The process view model, conforming to process view metamodel, is extracted from the application model using Application2ProcessView schema. The process view model is translated into process automata specific model using Process2Analysis and Process2Fc2 transformation rules. The desired integration property is typically a composition of formal relations and invariants over model elements. Framework uses analysis engine for reasoning the formal verification and MBVE for model manipulation and structural verification. Verification tool generates counter-example in case of any mismatch. Framework converts that feedback into a *process automata specific model*. The bidirectional *Process2Analysis* rules translate this to process view model to show the mismatch to the end-user.

5. Conclusion

We argued that enterprise application integration (EAI) is not just an interoperability issue that can be adequately addressed using plumbing techniques only, and the challenge lies in coming up with a pragmatic approach for ensuring semantically correct integration. We argued that though several formalisms, namely, Petri-nets, finite state automata, CSP etc provide the necessary rigor for analyzing properties of interest,

they however do not easily scale up and are a bit cumbersome to use for an average IT professional. We presented a model-driven bi-directional bridge between easy-to-use modeling notations and rigorous formal abstractions to analyze properties of interest such as compatibility and correctness (of business process integration) which, we believe, will help an average IT professional. We described a model-driven framework that combines the convenience of high level notations used by the industry practice and the rigor of formal verification techniques to automate the proposed approach. A holistic approach to EAI that combines integration of data, service and process models is our objective. Although the formal analysis described here is restricted to process view integration, the core concepts of our approach are designed to address EAI problem as a whole. We think the work described in this paper has helped us get an insight which will help in addressing the EAI problem comprehensively in future.

6. References

[1] Aalst W., The Application of Petri Nets to Workflow Management, *The Journal of Circuits, Systems and Computers*, 8(1), pp. 21–66, 1998.

[2] Abadi M. and Lamport L., The existence of refinement mappings. *Theoretical Computer Science* Vol: 82, pp 253-284, May, 1991.

[3] Ankolekar A., et al. DAML-S: Web Service Description for the Semantic Web, *Proceedings of the International Semantic Web Conference (ISWC)*, pp. 348–363, 2002.

[4] Barat S., Kulkarni V. and Janakiram D., A safety criterion for reusing a business process in the desired integrated process, *IEEE International Conference on Services Computing (SCC'06)*, pp. 381-389, 2006

[5] Barat S., Kulkarni V. and Janakiram D., A framework for business process integration, *Technical Report (IITM-CSE-DOS-2007-03) Indian Institute of Technology, Madras*, http://dos.cs.iitm.ernet.in /publications/techrep/2007.html, 2007.

[6] Berry G., The Esterel v5 Language Primer Version v5 91, http://www.esterel-technologies.com, 2000.

[7] Bouali A., Ressouche A., Roy V., and Simone R., The fc2tools Set, *8th International Conference on Computer Aided Verification*, LNCS: Vol 1102, 1996.

[8] Foster H., Uchitel S., Magee J. and Kramer J., Compatibility Verification for Web Service Choreography, *Proceedings of the ICWS IEEE*, pp. 738-741, 2004.

[9] Hamadi R. and Benatallah B., A Petri Net-based Model for Web Service Composition, *Conferences in Research and Practice in Information Technology Series*, Proceedings of the Fourteenth Australasian database conference on Database technologies, Volume 17, 2003.

[10] Hoare C. A. R. and Roscoe A. W., A Theory of Communicating Sequential Processes, *J. ACM* 31 (3), pp. 560-599, 1984.

[11] Hopcroft J.E., Motwani, R., and Ullman, J.D., Introduction to automata theory, languages, and computation, *Addison-Wesley*, 2001

[12] IBM, Draft UML 1.4 profile for automated business process with a mapping to BPEL1.0, *http://www-128.ibm.com/developerworks/rational/library/4593.html*, 2003.

[13] IBM, Specification: Business Process Execution Language for Web Services Version 1.1, *http://www 128.ibm.com/developerworks/library/specification/ws-bpel*, 2002.

[14] Johannesson P., Wangler B, and Jayaweera P, Application and Process Integration –Concepts, Issues, and Research Directions, *Information Systems Engineering Symposium CAiSE*, Springer Verlag, 2000.

[15] Kulkarni V. and Reddy S, Integrating Aspects with Model Driven Software Development, *Software Engineering Research and Practice*, pp 186-197, 2003.

[16] Linthicum D., Enterprise Application Integration, *Addison-Wesley*, 2000.

[17] Milner R., Communication and mobile systems: the Pi-calculus, *Cambridge University Press*, 1999.

[18] Moller A. and Klarlund N., MONA Version 1.4 User Manual, *Basic Research in Computer Science*, Information and computation 121(2), 1995.

[19] Muller, S, Xu, K., and Liu, Y., A static compliance-checking framework for business process models. *IBM System. Journal, 46, 2*, pp. 335-361, 2007.

[20] OMG, Model Driven Architecture, *http://www.omg.org/mda*.

[21] OMG, MOF Model to Text Transformation Language, *QVT RFP*, http://www.omg.org/docs/ad/06-04-03.pdf, 2006.

[22] OMG, Meta Object Facility (MOF) Specification, *OMG Document*, 1997.

[23] OMG, UML 2.0 OCL Specification, *OMG Document*, 2003.

[24] QVT Merge Group, MOF 2.0 Query/View/Transformation, 2005.

[25] Ring K. and Neil Ward-Dutton, White paper: Enterprise Application Integration: Making the Right Connections, *Ovum consulting group*, May, 1999.

[26] Wombacher A., Fankhauser P., Mahleko B. and Neuhold E., Matchmaking for Business Processes Based on Choreographies, *International Journal of Web Services*, 1(4):14-32, 2004.

Model-Based Test Complexity Analysis For Software Installation Testing

Jerry Gao, Karen Kwok
San Jose State University

Todd Fitch, Intuit Corp.
Intuit Corp.

ABSTRACT
Software testing is the last critical phase in software quality control. Software installation testing is one of the most important and complex tasks in system testing. However, in past years, researchers have not paid much attention to the related issues and challenges in software installation testing. One of them is test complexity analysis and planning. The paper uses a test model, known as *a semantic tree*, to assist engineers in modeling test complexity of software installation in terms of diverse system environments and configurations, various running conditions, and system functional features. The paper presents one systematic method based on the model to compute and analyze test complexity of software installation testing in three perspectives: system configurations, system running conditions, and system installation functions. The related application examples and experimental results are reported.

KEYWORDS
Software modeling and analysis for testing, software installation testing, and model-based testing complexity analysis.

1. INTRODUCTION

Software testing is the last critical phase in software quality assurance. Software installation testing is one of the important types of system testing. However, in past years, researchers have not made enough effort to tackle the related issues and challenges in software installation testing. Until now test and QA engineers lack well-defined installation test models, methods, and automation tools. In the real world, QA and test engineers always encounter three challenge issues in software installation validation.

1. A software product usually can be installed on a diverse system environment with many different configurations. Where is the solution, which helps them analyze diverse system configurations for installation testing?
2. A software product is required to be installed successfully under various system running conditions. Where is the solution, which helps them analyze various system running conditions for installation testing?
3. What is software installation test complexity for a given software product?
4. How can engineers use a systematic approach to identify a cost-effective test strategy (or test sequences) during test they planning?

This paper focuses on the first three issues and presents model-based approaches to addressing them. This paper uses a test model, known as a semantic tree [19], to demonstrate how to perform test complexity analysis of software installation testing in three aspects: a) system configurations, b) system running conditions, and c) system installation functions. The major contribution of this paper is its model-based test complexity analysis method and algorithms. Moreover, its application examples and case study results indicate that the proposed approach provides a systematic way to test modeling and test complexity analysis for software installation testing.

This paper is structured as follows. The next section discusses the background and related work in software installation testing and model-based testing. Section 3 reviews the semantic tree model, and its semantic spanning tree concepts and the related algorithms. Section 4 demonstrates how to use this model to compute the test complexity of software installation validation. The detailed computation formulas and algorithms are given for test complexity analysis. Moreover, application examples and statistic results of analyzing software installation test complexity for Turbo Tax. Finally, the concluding remarks and future work are mentioned in Section 5.

2. BACKGROUND AND RELATED WORK

What is software installation testing? According to [19], its major purpose is to validate the given software product to see if it can be correctly installed in a specified system environment with proper system configurations and running conditions. The major focus of software installation validation is to find the answers to the following questions:

- Can the software be properly installed on all specified system configurations?
- On the specified system configuration environment, can the software be successfully installed under each of the validated running conditions?
- Does the software demonstrate that its installation functions and behaviors behave correctly?

A detailed software installation test process has been given in [19]. As pointed in [19], the problem space of ***software installation testing*** can be presented as a 3-dimensional space, in which the X-axis presents all specified system configurations, the Y-axis presents all of system running conditions, and the Z-axis presents system installation functions. Clearly, well-defined test models and systematic methods are needed to model and present the issues in this space. This paper is written to address the mentioned software installation issues using a model-based approach.

Based on our recent literature survey, we found some related work that has contributed to software installation testing. For example, Edward Kit in his book [1] discussed the current status and existing problems in installation testing in the real world. In [3], Mark Pawson introduced the Install Shield Test Matrix, which can be useful for testers to identify, document, and select the major focuses in software installation validation. Using this matrix, a tester can select test items and design test cases. However, there are two issues with this matrix. First, there is a lack of detailed engineering guidelines and systematic solutions to help engineers identify and create this matrix. Second, if software with complex configurations can be executed under diverse running conditions to support different installation

functions, generating this matrix manually becomes very complex and too tedious. In addition, testers also need a rational approach and strategy to make cost-effective testing trade-off, measure test complexity, and analyze test cost and coverage for software installation. Furthermore, there are some challenges and methods in the development of software patches. The proposed solutions in [4] are useful to identify the problems, develop a patch, and create a deployable package. However, systematic methods are needed in software installation and patch testing in the industry to help engineers to identify and control the test coverage and selections of reusable test cases for installation testing in software evolution.

In 1999, Edward Kit in his book [1] reported the current practice status and the existing problems in software installation testing in the industry. Later, Chris Agruss [2] points out the test automation needs in software installation. Mark Pawson presents his approach – a test matrix to track test focuses and a series of test cases for software installation testing.

Recently, the model-based software testing is becoming a hot research topic in software engineering. There are numerous published papers addressing different model-based software test topics. For example, the basic concepts, motivations and issues of model-based software testing are discussed in [5][6][7][8]. Different models have used in model-based software testing, including state-based models [9][10], UML-based models [11][12], syntax grammar models[13],and the statistical models [14]. In [25], we introduce a semantic tree model for software installation testing, and discuss its model-based test criteria for system configurations, running conditions, and installation functions.

Until now, the model-based approach has been used in different areas of software testing. One area is model-based test generation. For example, Offutt et al. in [10][15] discuss how to generate tests from state-based model and UML-based model. Fujiwara et al, in [16] presents the test selection methods based on finite state models. Farchi et al in [17] share their thoughts on how to generate tests for standard conformance using a model-based approach.

In addition, there are a number of papers focusing on model-based test coverage analysis. Gao et al. in [19] discuss component-based test coverage analysis using component API-based function access models. D. Williams in [18] presents his approach to analyze test coverage based on functional faults and Failure Mode Effect Analysis (*FMEA*), which is useful for IC testing. *FMEA* approach performs a risk analysis on each component of a system.

Some researchers have used model-based approach to conduct regression testing. For example, Bogdan Korel et al [21] presented their technique to use an extended finite state machine (EFSM) as a model to support software regression testing. They used an automatic approach to identify model changes and impacts as well as related test changes and impacts.

Moreover, some recent publications use a model-based approach to validate non-functional requirements of software, such as reliability and performance. For example, Kirk Sayre and Jesse Poore [21] evaluated the system reliability using a model-based approach and metrics. Mahnaz Shamms et al. [22] presented a model-based approach for testing the performance of web applications. In this approach, they used EFSMs and data dependence models to measure system user response times and performance.

This paper focuses on test complexity measurement for software installation testing. We use a model-based approach to analyzing software installation test complexity in three aspects: system configurations, running conditions, installation functions. A model-based algorithm is provided to allow engineers to analyze installation test complexity in a systematic way.

3. A TEST MODEL FOR INSTALLATION TESTING

This section reviews the test model, known as the semantic tree model in [19] for software installation. Test engineers can use this model to analyze, model, and present diverse system configurations, complex running conditions, and various system installation functions during installation test planning.

In [19], we have proposed a model, known as a semantic tree model, which is formally defined as 3-tuple = (N, E, R), where

- N is a set of tree nodes. Three types of nodes exist: a) a single root node, b) intermediate nodes, and c) leaf nodes.
- E is a set of links in a tree. Each link connects a parent node and child node in a tree.
- R is a set of relations, and each item in R has a semantic label that presents one semantic relation between a parent node and its child nodes. Five types of semantic labels exist: OR, AND, NOT, NAND, and Select-1.

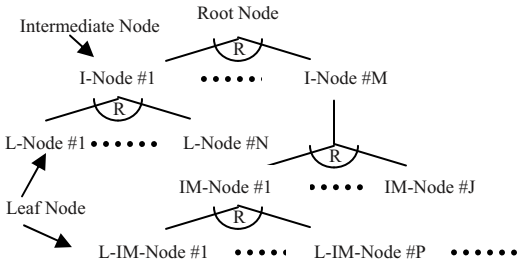


Figure 1 A Semantic Tree Model

Figure 1 shows an example of the generic semantic tree model, and Figure 2 shows the notation of the five different semantic labels in a model. Table 1 lists the semantics of a semantic tree model. As shown in Figure 3, this model can be useful to model and present diverse system environment configurations, various running conditions, and installation functions respectively [19]. In the rest of this section, some examples are given to demonstrate its applications.

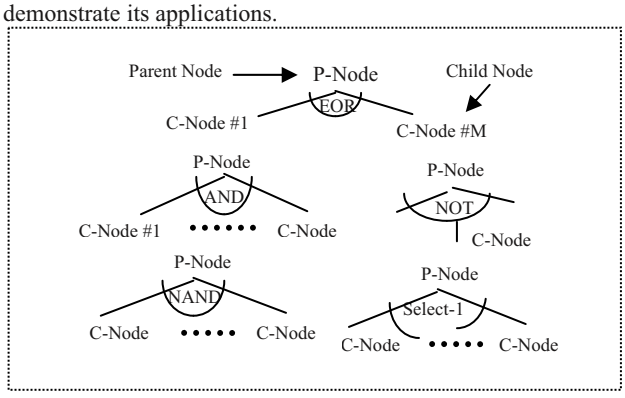


Figure 2 The Notations of the Five Semantic Labels

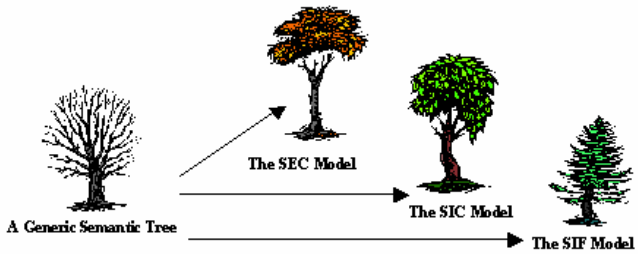


Figure 3 Semantic Tree Models for Software Installation

Table 1 The Semantics of A Semantic Tree Model

Relation	The semantics of different relations
EOR	It represents an Exclusive-OR relation between a parent node and its two child nodes. This indicates that only one of its child nodes can be selected to associate with its parent node.
AND	It represents an AND relation between a parent node and its child nodes. This indicates that all of its child nodes associate with its parent node at the same time.
SELECT-1	It represents a SELECT-1 relation with a parent node and its child nodes. This indicates that only one of its child nodes can be selected to associate with this parent node at any time.
NAND	It represents a NAND relation between a parent node and its child nodes. This indicates that all of its child nodes are not selected to associate with this parent node.
NOT	It represents a NOT relation between a parent node and its only child node. This indicates that the only child node is not selected to associate with this parent node.

A ***semantic spanning tree*** G_{SPT} is a sub-tree of a given semantic tree model G_{ST}, where it holds the following **properties**:

- G_{SPT} must include all parent nodes in G_{ST}.
- For each parent node Npi with the AND (or NAND) relation, G_{SPT} must include all of its child nodes and their corresponding links.
- For each parent node Npi with the EOR relation, G_{SPT} must include only one of its child nodes and its corresponding link.
- For each parent node Npi with the Select-1 relation, G_{SPT} must include only one of its child nodes and its corresponding link.
- For each parent node Npi with the NOT relation, G_{SPT} must include the only child node and its corresponding link.

Figure 4 shows a sample semantic tree model and two of its semantic spanning trees. The derivation procedure for generating a semantic spanning tree for the semantic tree G_{ST} is given in [25]. The detailed algorithm to generate all semantic spanning trees for a given semantic tree model is given in [24].

4. MODEL-BASED TEST COMPLEXITY ANALYSIS

As discussed in Section 2, the software installation testing problem space can be viewed in a 3-dimentional problem space. We can use the semantic tree model to model the test problem space and analyze its test complexity from three perspectives: a) system environment configurations, b) running conditions, and c) installation functions. In this section, we use three derived test models based on the semantic tree model to analyze the test complexity in each dimension.

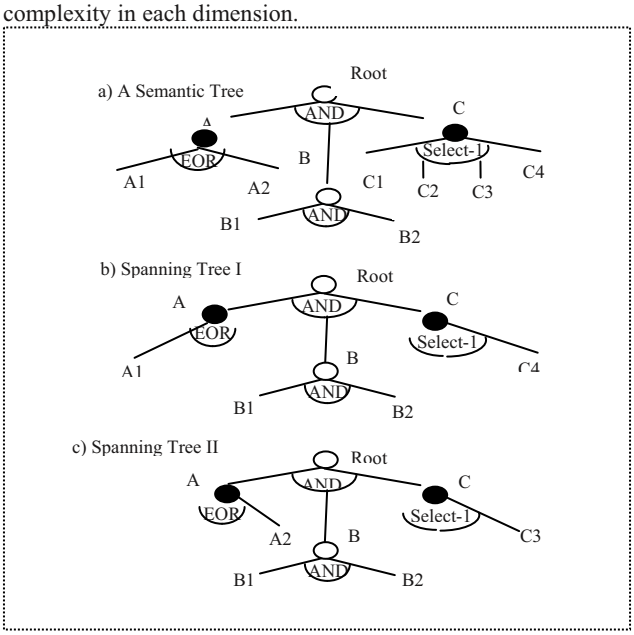


Figure 4 A Semantic Tree Model and Its Spanning Tree

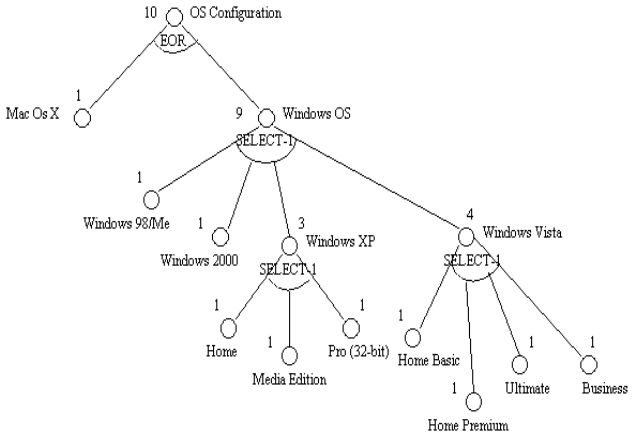


Figure 5 A SEC Model and Its Test Complexity for Turbo Tax

4.1 Test Complexity Analysis for System Configurations

As discussed in [25], a semantic tree model can be used to present diverse system environments and configurations using a hierarchical way. Let's use $G_{SEC} = (N_{SEC}, E_{SEC}, R_{SEC})$ to present a such model, where N_{SEC} is a set of nodes, E_{SEC} stands for a set of links between nodes, and R_{SEC} stands for a set of semantic relations between parent nodes and its child nodes. It is known as the System Environment Configuration (SEC) model, in which each leaf node presents one part (or component) of a configured software system or one type of its configurations. As shown in Figure 2, there may be five types of relations between a parent node and its child nodes in each SEC model. A parent node with an EOR relation suggests that only one of its two child nodes can be picked as its configuration. A parent node with a Select-1 relation indicates that any of its child nodes can be selected as one of its configuration. A parent node with an AND relation indicates that all its child nodes are required as a

part of its configuration. A parent node with a NOT relation suggest that its child node is not included in its configuration. A parent node with a NAND relation indicates that all its child nodes are not included in its configuration. Table 2 shows the detailed semantics of the five relations in a SEC model. Figure 5 shows an example of a SEC model, which presents all operating system configurations for Turbo Tax.

Clearly, a SEC model presents all the possible configurations for installation software because each of its spanning trees represents one of system configurations. Hence, as pointed out in [25], we must check the test coverage for each system environment configuration. In other words, we must test installation software under each system configuration.

Table 2 Semantic Relations in a SEC Model

Relations	Semantics in a System Environment Configuration Model
EOR	P-Node must be provided and set up with only one of its exclusive parts, which are denoted as two child nodes. In other words, the two parts can't be set up at same time.
AND	P-Node must be provided only when all of its child nodes are set up.
NOT	P-Node must be provided without setting up its specific part, denoted as the only child node.
NAND	P-Node must be provided without the support of some parts, denoted as its child nodes.
Select-1	P-Node can be set up with any of one of its child nodes.

Therefore, the test complexity of system environment configurations, denoted as $SEC_{Complexity}$, can be measured as follows.

$SEC_{Complexity}$ = No. of semantic spanning trees in G_{SEC}

The algorithm of generating a semantic spanning tree is given in [25]. The detailed algorithm of generating all spanning trees in a semantic tree is given in [24].

Here we present a model-based method and its algorithm to compute the test complexity of a SEC model in a hierarchical way when we consider the test complexity of each leaf node in G_{SEC} is 1. This suggests that one test script is needed to set up this configuration. To compute the test complexity of a SEC model G_{SEC}, we can work from leaf nodes to its parent nodes, until the root node in a hierarchical approach using the formula defined below.

For any parent node Npi of a semantic tree model G (say G_{SEC}) its test complexity can be computed based on its relation with its child nodes and the test complexity of its child nodes. Let Cj stands for a child node of Npi.

- If its semantic relation with its child nodes is SELECT-1, then its complexity can be computed as follows.

  $$Npi's\ T_{Complexity} = \sum (Cj's\ T_{Complexity}) \quad (1)$$

  Where j = 1, …, n, n is the number of its child nodes, and Cj is a child node of Npi.
- If its semantic relation with its child nodes is EOR, then its complexity can be computed as follows.

  $$Npi's\ T_{Complexity} = \sum\ (Cj's\ T_{Complexity}) \quad (2)$$

  Where j = 1 or 2, and Cj is a child node of Npi.
- If its semantic relation with its child nodes is AND, then its complexity can be computed as follows.

  $$Npi's\ T_{Complexity} = \prod (Cj's\ T_{Complexity}) \quad (3)$$

  Where j = 1, …, m, m is the number of its child nodes, and Cj is a child node of Npi.
- If its semantic relation with its child nodes is NAND, then its complexity can be computed as follows.

  $$Npi's\ T_{Complexity} = 0 \quad (4)$$

  Where j = 1, …, m, m is the number of its child nodes, and Cj is a child node of Npi.

  According to the semantics of the NAND relation, all child nodes of *Npi* which are not supported in this release of software.
- If its semantic relation with its child node is NOT, then its test complexity should be 0. (5)

According to the semantics of the NOT relation, the child node refers to a configuration which is not required for a product release, hence the test complexity concerning this child node is 0. Figure 5 shows the configuration test complexity of the given SEC model for Turbo Tax. For the Windox XP node, it has three child nodes with SELECT-1 relation, its test complexity equals to 3, which is the summation of its child nodes' test complexity. According to Figure 5, the total configuration test complexity of the given SEC model is 10.

4.2 Test Complexity Analysis for System Running Conditions

Similarly, we can use a semantic tree model to present the diverse system installation conditions, which is known as the System Installation Condition (SIC) model. For a given SIC model, $G_{SIC} = (N_{SIC}, E_{SIC}, R_{SIC})$, where N_{SIC} is a set of nodes, E_{SIC} stands for a set of links between nodes, and R_{SIC} stands for a set of semantic relations between parent nodes and its child nodes. In this model, the root node presents the overall condition under the system installation, and it depends on a number of conditions. Each condition, as a parent node, may depend on a number of sub-condition factors as its child nodes. Each leaf node presents a special condition of its parent node (as a condition factor). The same five semantic relations (AND, NAND, EOR, SELECT-1, and NOT) can be used to present five different types of relationships between a parent node and its child nodes in a SIC model. Table 3 provides the detailed semantics of the five relations in a SIC model. Figure 6 displays an example, which presents various running conditions of Turbo Tax. This model presents various system installation conditions in a hierarchical tree model.

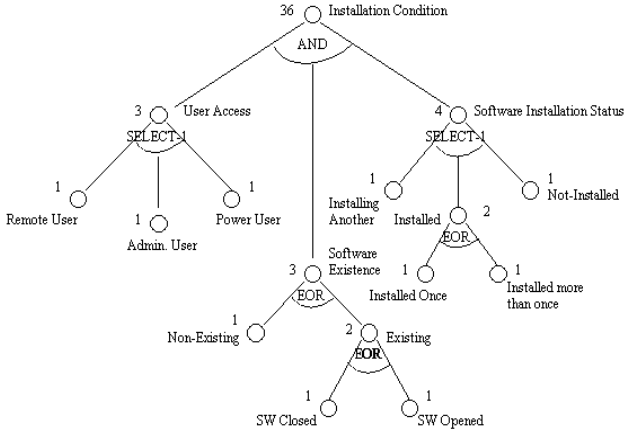


Figure 6 A SIC Model and Test Complexity for Turbo Tax

Clearly, a SIC model presents all the possible system installation conditions of a software product since each of its spanning trees represents one combinational installation condition.

Table 3 Semantic Relations in SIC Model

Relations	Semantics in an Installation Condition Model
EOR	P condition holds only when one of its two exclusive sub-conditions (denoted as child conditions) holds.
AND	P condition holds only when all of its child conditions hold.
NOT	P condition holds only when its child condition is not hold.
NAND	P condition holds only when all of its child conditions are not hold.
Select-1	P condition holds when any one of its child conditions holds.

```
int TestComplexity(node N)
{ // This function computes the test complexity of a given semantic tree
  // (or a sub-tree), which is a SEC model or a SIC model.
  // The node N is the root node of a given semantic tree (or a sub-tree).
  // The final return result of this function is the computed total
  // test complexity of the given semantic tree model.

int testComplexity = 0;
// this initialization only applies the first time this code is executed.

if node == leaf node then {
            node.testComplexity = 1;
            return node.testComplexity; }
else { // else node is a parent node, including a root node
            switch (node N's semantic relation) {
            case 'AND':
            for all child nodes, do
            node.testComplexity =
                  node.testComplexity * TestComplexity(child node);
            break;
            case 'EOR' or 'SELECT-1':
            for each child node, do
            node.testComplexity =
                  node.testComplexity + TestComplexity(child node);
            break;
            case 'NAND':
            node.testComplexity = 0;
            break;
            default 'NOT':
            node.testComplexity = 0;
            break;                    }
            return testComplexity;
    }
}
```

Figure 7 The Algorithm for Test Complexity Analysis for a SEC/SIC model

As discussed in [19], under any system configuration SECi, engineers must perform software installation testing for each combinational system running condition. As we know that it is a common practice to use one test script to set up one combinational running condition for a system. Hence, the test complexity ($SIC_{Complexity}$) of system installation conditions can be computed as follows:

$SIC_{Complexity}$ = No. of semantic spanning trees in G_{SIC}

Similar to $SEC_{Complexity}$, we can use the formulas (1)-(5) to compute $SIC_{Complexity}$ of a SIC model (G_{SIC}) in a hierarchical way from its leaf nodes to the root node. Figure 7 shows the proposed algorithm based on a given semantic model (a SEC model or a SIC model).

As shown in Figure 7, the node *Installed* has an EOR relation with its two child nodes: *Installed Once*, and *Installed More Than Once*. Its test complexity is equal to 2, which is the summation of the test complexity of its two child nodes. The root node of the SIC model has an AND relation with its three nodes. Its test complexity is equal to 36 based on the formula (3).

4.3 Test Complexity Analysis for System Installation Functions

The semantic tree model can also be used to present the system installation functions. We define it as the System Installation Function (SIF) model. Let's use $G_{SIF} = (N_{SIF}, E_{SIF}, R_{SIF})$ to present a SIF model, where N_{SIF} is a set of nodes, E_{SIF} stands for a set of edges between nodes, and R_{SIF} stands for a set of semantic relations between parent nodes and its child nodes. This model presents all system installation functions in a hierarchical format. A parent node in a SIF model presents a high-level function and its child nodes present its low-level functions (or sub-functions). A similar set of five semantic relations is used to represent the relations between a parent and its child nodes. Table 4 shows the detailed semantics of the five relations in a SIF model.

It should be noted that NAND and NOT relations only useful when some functions (or components) of the system are not required (supported) in a product release. Figure 9 displays a sample SIF model for a product (Turbo Tax). It presents its related system installation functions. Each lead node represents a low-level system installation function. Each parent node stands a high-level installation function. The SIF model can be used to present system functions in two different views. One is a hierarchical function view like Figure 9. And the other is a functional feature view, which presents a system functional feature in terms of its required system components (or sub-systems).

Table 4 Semantic Relations in a SIF Model

Relations	Semantics in a System Installation Function Model
EOR	The P function is supported only when any of its two exclusive sub-functions (denoted as child nodes) is provided.
AND	The P function is supported only when all of its sub-functions (as denoted child nodes) are provided.
NOT	The P function is supported without its specific sub-function, denoted as the only child node.
NAND	The P function is supported without the support of some of its sub-functions, denoted as its child nodes.
Select-1	P function is provided when anyone of its sub-functions (denoted as child nodes) is provided.

In [25], we have defined a system installation function test coverage criteria for a given software product P, executed under a system installation condition SICj in a configured system environment SECi below.

Leaf Node Function Test Criterion:

For any leaf node Ni in G_{SIF}, this criterion is achieved when the given T_{IS} includes at least one test case, which exercise the corresponding function of Ni.

Adequate Leaf Node Function Test Criterion:

For any leaf node Ni in G_{SIF}, this criterion is achieved when the given T_{IS} includes an adequate test set, which exercise the corresponding function of Ni.

Adequate Parent Node Function Test Criterion:

For any parent node Npi in G_{SIF}, including the root node and intermediate nodes, this criterion is achieved only when the given T_{IS} includes an adequate test set for each child node. In other words, all of its child nodes have achieved its adequate test criterion.

To achieve these test criteria, we need to analyze the test complexity for testing installation functions. Since a spanning tree of a SIF model has no meaning in testing of software installation, engineers need a systematic way to measure and predicate the test complexity of software installation functions.

In this paper, we provide a well-defined model-based approach, which computes the test complexity of installation function testing based on a SIF model.

In a SIF model G_{SIF}, any leaf node *Ni* presents a bottom-level software installation function. Hence, it must have an adequate functional test set. Let's use the number of test cases (or scripts) of its test set as its test complexity. We can compute the test complexity of a SIF model from leaf nodes to a root node in a hierarchical way.

For any parent node Npi of G_{SIF}, its function test complexity can be computed based on its relation with its child nodes and their test complexity. Let Cj stands for a child node of Npi.

- If its semantic relation with its child nodes is SELECT-1, then its complexity can be computed as follows.

  $$Npi's\ FT_{Complexity} = \sum (Cj's\ FT_{Complexity}) \quad (6)$$

  Where j = 1, …, n, n is the number of its child nodes, and Cj is a child node of Npi.
- If its semantic relation with its child nodes is EOR, then its complexity can be computed as follows.

  $$Npi's\ FT_{Complexity} = \sum (Cj's\ FT_{Complexity}) \quad (7)$$

  Where j = 1 or 2, and Cj is a child node of Npi.
- If its semantic relation with its child nodes is AND, then its complexity can be computed as follows.

  $$Npi's\ FT_{Complexity} = \sum (Cj's\ FT_{Complexity}) \quad (8)$$

  Where j = 1, …, m, m is the number of its child nodes, and Cj is a child node of Npi.
- If its semantic relation with its child nodes is NAND, then its complexity can be computed as follows.

  $$Npi's\ FT_{Complexity} = 0 \quad (9)$$

  Where j = 1, …, m, m is the number of its child nodes, and Cj is a child node of Npi.

  According to the semantics of the NAND relation, all child nodes of *Npi* are its sub-functions which are not required for this released product for installation function testing.
- If its semantic relation with its child node is NOT, then its test complexity should be 0. According to the semantics of the NOT relation, the child node is not included as a part of the supporting functions, hence the test complexity concerning this child node is 0. (10)

It is clear that these formulas are similar to the previous formulas except formula (8). Since the installation functions for a product usually are independent, that is why the complexity of a parent with an AND relation can be computed using the formula in (8). Figure 8 displays the test complexity for each leaf node with the number of function test cases. Its function test complexity is 39.

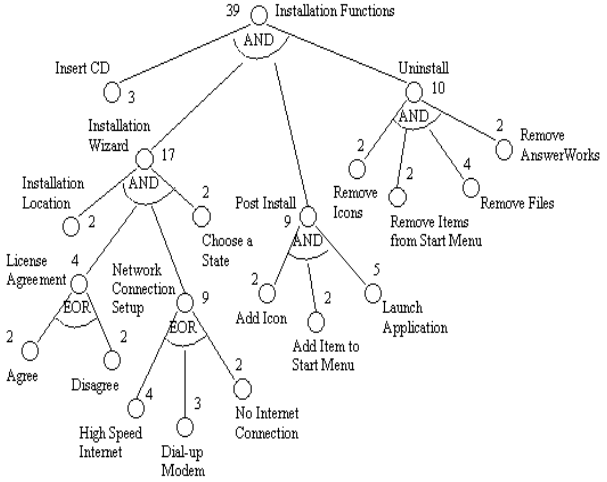


Figure 8 A SIF Model and Its Test Complexity for Turbo Tax

We have used the proposed model-based method in test complexity analysis for Turbo Tax. We create three semantic tree models for Turbo Tax:

- A SEC model for Turbo Tax, which presents its diverse system configurations.
- A SIC model for Turbo Tax, which presents its various system running conditions.
- A SIF model for Turbo Tax, which presents its installation functions.

Table 5 shows the statistic analysis results of three models (SEC. SIC, and SIF) for Turbo Tax. It displays the detailed information about these models, including its test complexity and height, the number of spanning trees, the number of leaf nodes, and the number of tree nodes. Clearly, the test complexity of the SEC model is the same as the number of its different semantic spanning trees. Similarly, the test complexity of the SIC model equals to the number of its semantic spanning trees. Since the semantic spanning trees of software installation testing of Turbo Tax, hence, we just compute its functional test complexity using the given algorithm listed in Figure 9. Its function test complexity is 36 for Turbo Tax considering the fact that these function features are independent.

Table 5 The Statistics of Three Models for Turbo Tax

Model	No. of Spanning Trees	No. of Nodes	No. of Leaf Nodes	Height of Semantic Tree Model	Test Complexity
SEC	10	14	10	4	10
SIC	36	16	10	4	36
SIF	---	21	15	3	39

5. CONCLUSION AND FUTURE WORK

This paper focuses on two problems in software installation testing. The first issue is how to measure and analyze the test complexity of software installation testing. This paper uses a semantic tree model proposed in [19] to demonstrate how to

measure software installation test complexity in three perspectives: system configurations, running conditions, and installation functions. The proposed approach has been used in a project at a local software company to help test engineers in software installation testing, and has received very positive feedbacks in modeling software installation test complexity and analysis. The reported case study results indicates that the proposed model-based approach has the distinct advantage on systematic test complexity analysis and measurement for software installation testing. The future work of this research work is to develop a model-based test complexity analysis and planning tool for software installation testing. It can be useful in software installation testing based on the proposed models to support test modeling and analysis, test generation and sequences, and test coverage analysis.

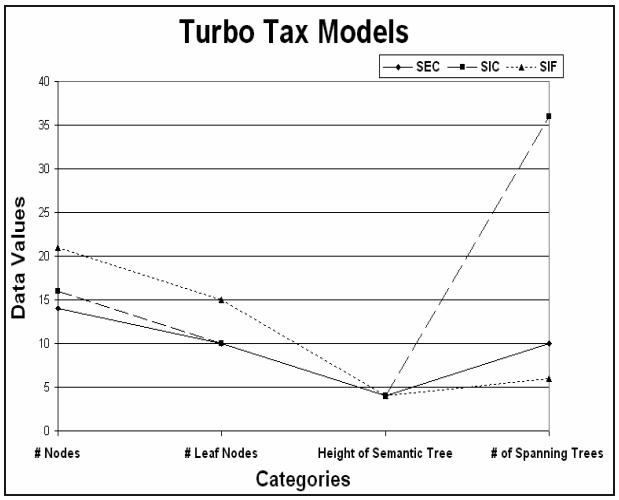


Figure 9 Statistic Results of three Models for Turbo Tax

6. REFERENCES

[1] Kit, E. (1999). Software Testing in the Real World: Improving the process (5th). England: ACM Press.

[2] Agruss, C. Software Installation Testing: How to automate tests for smooth system installation. *Testing & Quality Magazine, Vol. 2, Issue 4*. July/August 2000.

[3] Mark Pawson, "The Test Matrix: How one company kept a complex test on track?" (2001). Retrieved at URL: http://www.stickyminds.com/ March 3, 2006.

[4] Ruest, Nelson, A Practical Guide For patch testing, from http://www.wise.com/Library/Patch_Whitepaper.pdf

[5] Ibrahim K. EI-Far and James A. Whittaker, Model-based Software Testing, Encyclopedia on Software Engineering (edited by J.J. Marciniak), Wiley, 2001.

[6] Pretschner and Bruno Legeard, "A Taxionomy of Model-Based Testing by Mark Utting", Technical Report, 2006.

[7] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, and C. M. Lott. Model-based testing of a highly programmable system. In *Proceedings of the International Symposium on Software Reliability Engineering*, pages 174-179, 1998.

[8] S. R. Dalal, A. Jain, N. Karunanithi, "Model-Based Testing in Practice", Proceedings of International Conference on Software Engineering (ICSE'99), 1999.

[9] Avik Sinha and Carol Smidts, "HOTTest: A Model-Based Test Design Technique for Enhanced Testing of Domain-Specific Applications", ACM Transaction on Software Engineering and Methodology, Vol. 15, No. 3, July 2006, Pages: 242-278.

[10] J. Offutt, S. Liu, and A. Abdurazik. Generating test data from state-based specifications. *Journal of Software Testing, Verification & Reliability*. 13(1): 25-53, April 2003.

[11] Harry Robinson, "Graph Theory Techniques in Model-Based Testing", the proceedings of 1999 International Conference on Testing Computer Software, 1999.

[12] Mike Barnett, et al. "Scenario-oriented Modeling in AsmL and its Instrumentation for Testing", *2nd International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools* , May 2003.

[13] Mikhail Auguston, James Bret Michael, and Man-Tak Shing, "Environment Behavior Models for Scenario Generation and Testing Automation",

[14] J. A. Whittaker and M. G. Thomason, "A Markov Chain Model for Statistical Software Testing", IEEE Transactions on Software Engineering, V. 20, No. 10, pp. 812-824, October 1994.

[15] J. Offutt and A. Abdurazik, "Generating Tests from UML Specifications". Second International Conference on the Unified Modeling Language (UML99). Fort Collins, CO, October 1999.

[16] E. Farchi, A. Hartman, and S. S. Pinter, "Using a Model-Based Test Generator to Test for Standard Conformance", IBM Systems Journal. V. 41, No. 1, pp. 89-110, 2002.

[17] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test Selection Based on Finite State Models". IEEE Transactions on Software Engineering. V. 17, No. 6, pp. 591-603, June 1991.

[18] D. Williams, "Test Coverage Models for System Test?" Proceedings of International Test Conference, Oct. 2002.

[19] J. Gao, R. Espinoza, and Jingsha He, "Testing Coverage Analysis for Software Component Validation", Proceedings of 29th Annual International Conference on Computer Software and Applications, Volume 1, PP. 463-470, July 2005.

[20] Bogdan Korel, et al., "Model Based Regression Test Reduction Using Dependence Analysis", Proceedings of the International Conference on Software Maintenance (ICSM'02), 2002.

[21] Mahnaz Sharms, Diwakar Krishnamurthy, "A Model-Based Approach for Testing the Performance of Web Applications", Proceedings of the Third International Workshop on Software Quality Assurance (SOQUA'06), 2006.

[22] Kirk Sayre and Jesse Poore, "A Reliability Estimator for Model Based Software Testing", Proceedings of the 13th International Symposium Software Reliability Engineering (ISSRE'02), 2002.

[23] Pretschner, et al., "One Evaluation of Model-Based Testing and Its Automation", Proceedings of 27th International Conferences on Software Engineering (ICSE2005), St. Louis, MO, USA, 2005.

[24] Karen Kwok, "A Model-Based and Test Analysis Methodology for Software Installation Testing", Master Thesis, San Jose State University, August 2007.

[25] Jerry Gao, et al, "Toward Test Modeling and Analysis for Installation Testing", Proceedings of SEKE2007, Boston, USA, July, 2007.

A Similarity Analysis Model for Semantic Web Information Filtering Applications

Lucas Drumond, Rosario Girardi, and Fabio Silva
UFMA – Federal University of Maranhão
Av. dos Portugueses, s/n
São Luís, Brazil

Abstract

This work proposes a similarity analysis model for content-based filtering based on the technology of the Semantic Web. Ontologies are the standard Semantic Web knowledge representation formalism. Thus, the proposed structure for representing user models and information items is based on ontologies. The similarity between such structures is calculated based on the similarity between concepts of an ontology. A similarity measure is presented and formalized according to a description logic. An experiment conducted for its preliminary evaluation is also introduced.

1. Introduction

Through the Web, a large amount of services and information is available to users. Usually, Web users have information needs that can be satisfied by a restricted set of Web documents. Identifying such documents manually in the huge amount of currently available documents is a human impossible task. This phenomenon, in which the user has access to an amount of information larger than that it is able to deal with, is known as information overload.

This scenario has created a demand for effective methods for accessing information. One approach to the information overload problem is information filtering [3], a research line that study how to satisfy the information needs of heterogenous users through dynamic and unstructured information sources.

The first generation of the Web was developed focusing on how the information is displayed and not on how it is structured. The lack of semantic structure of the information available on the Web affects the filtering effectiveness. Differently, in the Semantic Web [17], data are structured in a way that can be "understood" by software applications such as intelligent agents. This semantic structure can be used to increase the effectiveness of information retrieval and filtering systems.

Similarity is a very important concept in this context. An item is recommended to a user if it is similar to its interests. On way of taking advantage of the Semantic Web technologies in information filtering and retrieval applications is using measures to compute the similarity considering ontologies, one of the formalisms used to represent Semantic Web resources.

This paper introduces a measure to compute the similarity between the concepts of an ontology hierarchy, and a semantic case based similarity model which uses such a measure.

The paper is organized as follows. Section 2 introduces the semantic case based similarity model. Section 3 describes an experiment carried out to its preliminary evaluation. Section 4 analyses related work on similarity measures. Finally, Section 5 concludes this paper with a discussion of the results obtained in this work and some remarks on further work.

2. The Proposed Similarity Analysis Model

The similarity model proposed in this paper is an adaptation from similarity between semantic cases proposed on [10] and [11], adapted to the information filtering context and using ontologies.

The first step in order to apply this model is to identify the semantic cases of an application domain. Semantic cases are user groups of interests, in which information items can be classified. For instance, a group of interest may be *sports* whose values can be football, tennis, swimming, etc.

After the identification of semantic cases, a hierarchy of categories associated to each semantic case is built. The categories of each hierarchy are the terms or values that each semantic case may assume. Each semantic case is represented as the root of its own hierarchy that can be constructed with the aid of a domain specialist or borrowed from an available ontology.

User models are composed by a set of semantic cases, representing the categories of user interests and a set of concepts for each semantic case, representing the specific interests of a user in each category. An information item model is organized in a similar way.

Finally, the similarity between a user model and an information item is calculated as follows: for each semantic case that appears in both the models, the similarity among the values of the semantic cases associated to the user and the ones associated to the information item is calculated.

2.1. Semantic Cases

A semantic case represents a characteristic of an information item through which user´s interests can be specified. A hierarchy of terms is associated with each semantic case. These terms are the possible values that a user model or information item model can have for each semantic case. For instance, in the context of the Infonorma system [7][8][9], a legal recommender system, semantics cases would represent a legal branch and the type of a normative instrument, main groups of interest of legal users.

Consider, in another example, a system of a hypothetical supermarket with the goal of recommending products for its customers. Such products can be divided in meats, vegetables, or masses. Meats, for instance, can be classified according to the hierarchy shown in Figure 1. Thus, the

semantics cases of interests are meat, vegetables and masses, and the types of each one are the terms of each case.

Formally, we can represent a semantic case as a concept C in a T-Box $\mathcal{T}$. The terms of semantic case C are all the concepts of $\mathcal{T}$ that are subsumed by C, that is, a concept D is a term of semantic case C if, and only if, $C \sqsupseteq D$. In the example of the supermarket, the concept *Meat* represents a semantic case and its subclasses are the values that the semantic case *Meat* can assume.

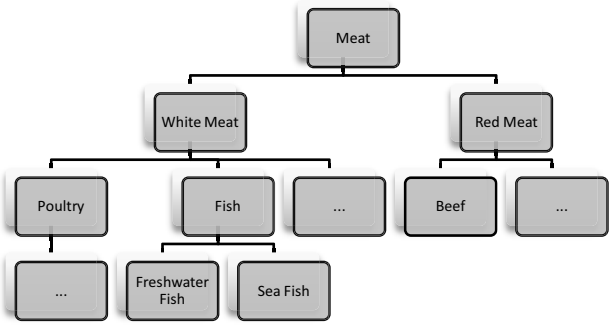


Figure 1 - Example of a semantic case hierarchy

2.2. User and Information Items Models

It was said that semantic cases represent categories of interests of the users and that such categories can be used to describe an information item. In this way, user and information item models must reference the values of the semantic cases.

Both users and information item models are represented by a set of values for each considered semantic case. These values are terms or concepts in the ontology hierarchy associated with the semantic case. Each semantic case in a user or information item model has a weight that represents its relevance in the application domain.

In the similarity analysis model considered in this work, a user model, denoted by $\mathcal{U}$, can be defined as follows:

$$\mathcal{U} = \{i | i \text{ is a semantic case that represents a group of interests of the } \mathcal{U} \text{ user } \}$$

A weight is assigned to each semantic case, denoted by w_i meaning how interested the user is in the group of interest i ($0 < w_i \leq 1$). For each semantic case i there is $\mathcal{U}_i$, the set of the values of i in which the $\mathcal{U}$ user has interest.

$$\mathcal{U}_i = \{j | j \text{ is a term of semantic case } i\}$$

The representation of the information items must be coherent with the representation of the user models. Here, a representation of an information item is denoted by $\mathcal{E}$:

$$\mathcal{E} = \{i | i \text{ is a semantic case used to describe the } \mathcal{E} \text{ information item}\}$$

Similarly, each semantic case describing an information item is associated with a set of terms:

$$\mathcal{E}_i = \{j | j \text{ is a term of the semantic case } i\}$$

2.3. Similarity between Semantic Cases

2.3.1. Properties of similarity measures

The concept of similarity is very intuitive and widely used. Some intuitive notions of this concept are discussed in [13]. Two items are considered similar regarding the amount of commonalities they share. The more characteristics they have in common, the higher similarity between them. On the other hand, the more differences they have, the less similar they will be. The higher similarity between two items is achieved when they are identical.

Besides the empirical criteria, many formal models have been proposed to express the similarity. Some of the premises assumed by formal models of a similarity relationship are listed in [18]:

- **Reflexivity**: An object is similar to itself;
- **Symmetry**: If the A object is similar to the B object then B is also similar to A.

The proposed similarity measure was developed in such a way that it holds the symmetry and reflexivity properties having value range between 0 and 1.

Be $\mathcal{U}$ and $\mathcal{E}$ two semantic case based models and $similarity(\mathcal{U}, \mathcal{E})$ a function that denotes the similarity between these two sets, then $similarity(\mathcal{U}, \mathcal{E})$ must hold the following properties. First, the similarity between a semantic case based model and itself should be maximum and equal to 1. This property is related to reflexivity. Formally:

$$similarity(\mathcal{U}, \mathcal{U}) = 1 \quad (1)$$

The similarity between two models that share no commonalities, i.e. two models which elements have no similarity between them, should be minimal and equal to 0. As property 1 states that the maximum similarity is 1, the similarity between two arbitrary models should be:

$$0 \leq similarity(\mathcal{U}, \mathcal{E}) \leq 1 \quad (2)$$

Also, the similarity function should be symmetric. Intuitively it can be stated that if A is similar to B, then B is similar to A in the same extent. Therefore,

$$similarity(\mathcal{U}, \mathcal{E}) = similarity(\mathcal{E}, \mathcal{U}) \quad (3)$$

2.3.2. The proposed similarity measures

As stated before, the similarity between two semantic case based models is given as a function of the values of the semantic cases in each model. Such similarity is computed adding, for each semantic case i appearing both in $\mathcal{U}$ and in $\mathcal{E}$, the higher value of the similarity between the elements of $\mathcal{U}_i$ and $\mathcal{E}_i$ times the weight of i. This weight can be the one in $\mathcal{U}$ or in $\mathcal{E}$ or a combination of both. Other criteria could be used to the similarity between the set of semantic cases as, for instance, the mean of the similarities

between $\mathcal{U}_i$ and $\mathcal{E}_i$. However, the higher value approach was the one that fit best in properties 1, 2 and 3.

The $similarity(\mathcal{U}, \mathcal{E})$ is given by:

$$similarity(\mathcal{U}, \mathcal{E}) = \sum_{\forall i \in \mathcal{U} \cap \mathcal{E}, \forall j \in \mathcal{U}_i, \forall k \in \mathcal{E}_i} w_i \cdot \max sim(\mathcal{U}_{ij}, \mathcal{E}_{ik}) \quad (4)$$

where:

a) $\mathcal{U}_{ij}$ is the term "j" associated to the semantic case i in the user model $\mathcal{U}$;
b) $\mathcal{E}_{ik}$ is the term "k" associated to the semantic case i in the information item model $\mathcal{E}$;
c) w_i is the weight of the semantic case i;

In order for the function $similarity(\mathcal{U}, \mathcal{E})$ to follow properties 1, 2 and 3, the function $sim(\mathcal{U}_{ij}, \mathcal{E}_{ik})$ must also follow these properties and the following condition must be true:

$$\sum_{\forall i \in \mathcal{U} \cap \mathcal{E}} w_i = 1 \quad (5)$$

In order to prove that the $similarity(\mathcal{U}, \mathcal{E})$ function follows the properties 1, 2 and 3 it is assumed now that $sim(\mathcal{U}_{ij}, \mathcal{E}_{ik})$ also follows those properties. This assumption will be proved further. Thus, the reflexivity property is granted because in the computation of $similarity(\mathcal{U}, \mathcal{U})$, equal terms are compared and, as it was assumed that $sim(\mathcal{U}_{ij}, \mathcal{E}_{ik})$ follows properties 1 and 2, then, the similarities between these equal terms will be 1. Hence, $\max sim(\mathcal{U}_{ij}, \mathcal{U}_{ik})$ will allways be 1 and:

$$similaridade(\mathcal{U}, \mathcal{U}) = \sum_{\forall i \in \mathcal{U} \cap \mathcal{E}, \forall j \in \mathcal{U}_i, \forall k \in \mathcal{E}_i} w_i \cdot \max sim(\mathcal{U}_{ij}, \mathcal{U}_{ik}) = \sum_{\forall i \in \mathcal{U} \cap \mathcal{E}, \forall j \in \mathcal{U}_i, \forall k \in \mathcal{E}_i} w_i \cdot 1 = \sum_{\forall i \in \mathcal{U} \cap \mathcal{E}} w_i = 1$$

Once assumed that $sim(\mathcal{U}_{ij}, \mathcal{E}_{ik})$ follows property 2, the higher value that $\max sim(\mathcal{U}_{ij}, \mathcal{E}_{ik})$ can assume is 1 and the lower, 0. As it has been shown that the function $similarity(\mathcal{U}, \mathcal{E})$ assumes the value 1 when $\mathcal{U} = \mathcal{E}$, to guarantee it follows property 2, all that it should be demonstrated is that this function has 0 as lowest value. For two sets $\mathcal{U}$ e $\mathcal{E}$ that do not share any commonality, $sim(\mathcal{U}_{ij}, \mathcal{E}_{ik}) = 0$ for every $(\mathcal{U}_{ij}, \mathcal{E}_{ik})$. Then,

$$similarity(\mathcal{U}, \mathcal{E}) = \sum_{\forall i \in \mathcal{U} \cap \mathcal{E}, \forall j \in \mathcal{U}_i, \forall k \in \mathcal{E}_i} w_i \cdot 0 = 0$$

The similarity function follows property 3 since $sim(\mathcal{U}_{ij}, \mathcal{E}_{ik}) = sim(\mathcal{E}_{ik}, \mathcal{U}_{ij})$.

$$similaridade(\mathcal{U}, \mathcal{E}) = \sum_{\forall i \in \mathcal{U} \cap \mathcal{E}, \forall j \in \mathcal{U}_i, \forall k \in \mathcal{E}_i} w_i \cdot \max sim(\mathcal{U}_{ij}, \mathcal{E}_{ik}) = \sum_{\forall i \in \mathcal{U} \cap \mathcal{E}, \forall j \in \mathcal{U}_i, \forall k \in \mathcal{E}_i} w_i \cdot \max sim(\mathcal{E}_{ik}, \mathcal{U}_{ij}) = similaridade(\mathcal{E}, \mathcal{U})$$

Before introducing the calculation of the closeness between two terms, it is worth to remember that a term is a concept of an ontology. Thus, a function for computing the closeness between ontology concepts can be used to compute the similarity between terms.

The closeness between two concepts C e D is computed considering the hierarchy of the respective semantic case. The set $C_{\sqsubseteq}$ denotes the set of all concepts E that subsume C ($E \sqsupseteq C$). For instance, considering Figure 1, it is true that:

$$(Sea\ Fish)_{\sqsubseteq} = \{Sea\ Fish, Fish, White\ Meat, Meat\}$$
$$(Beef)_{\sqsubseteq} = \{Beef, Red\ Meat, Meat\}$$
$$(Meat)_{\sqsubseteq} = \{Meat\}$$

It is important to stress that, when comparing two objects, one is interested in determining how much commonality they share. Within an ontology, if $E \sqsupseteq C$ and $E \sqsupseteq D$ then the concepts C and D share all characteristics of the concept E. This way, the set of all concepts which characteristics are present both in concepts C and D is given by $C_{\sqsubseteq} \cap D_{\sqsubseteq}$. Intuitively, one can realize that the higher the value of $|C_{\sqsubseteq} \cap D_{\sqsubseteq}|$ (number of elements of set $C_{\sqsubseteq} \cap D_{\sqsubseteq}$), the higher the similarity between C and D.

However, unless the two concepts being compared are equivalent, the characteristics shared by both of them describe only a part of each one. It is also needed to take into account the characteristics that the concepts do not share. An approach for that is to define the similarity between two concepts as the proportion of the amount of characteristics they have in common and all the characteristics of each one. As mentioned before, $|C_{\sqsubseteq} \cap D_{\sqsubseteq}|$ is an approximation for the amount of characteristics both concepts have in common. Similarly, an approximation for the amount of characteristics in the concepts C and D may be defined, respectively, as $|C_{\sqsubseteq}|$ and $|D_{\sqsubseteq}|$. The similarity between two concepts is a function defined by:

$$sim(C, D) = \frac{2 \cdot |C_{\sqsubseteq} \cap D_{\sqsubseteq}|}{|C_{\sqsubseteq}| + |D_{\sqsubseteq}|} \quad (6)$$

Considering the hierarchy of Figure 1, it is possible to compute the similarity between the concepts *Sea Fish* (represented as SF) and *Freshwater Fish* (represented as FF), given by:

$$sim(SF, FF) = \frac{2 \cdot |SF_{\sqsubseteq} \cap FF_{\sqsubseteq}|}{|SF_{\sqsubseteq}| + |FF_{\sqsubseteq}|} = \frac{2 \cdot 3}{4 + 4} = \frac{6}{8} = 0.75$$

The similarity between the concepts *Freshwater Fish* (FF) and *Poultry* (P) is:

$$sim(FF, P) = \frac{2 \cdot |FF_{\sqsubseteq} \cap P_{\sqsubseteq}|}{|FF_{\sqsubseteq}| + |P_{\sqsubseteq}|} = \frac{2 \cdot 2}{4 + 3} = \frac{4}{7} = 0.5714$$

Earlier it was assumed that the function $sim(C, D)$ follows the three properties identified in Section 2.3.1. Now it will be demonstrated that the assumption is true. In first place, $sim(C, D)$ reaches its higher value (which is 1) when $C \equiv D$, because in this case: $C_{\sqsubseteq} \cap D_{\sqsubseteq} = C_{\sqsubseteq} \cap C_{\sqsubseteq} = C_{\sqsubseteq}$, and then:

$$sim(C, C) = \frac{2 \cdot |C_{\sqsubseteq} \cap C_{\sqsubseteq}|}{|C_{\sqsubseteq}| + |C_{\sqsubseteq}|} = \frac{2 \cdot |C_{\sqsubseteq}|}{|C_{\sqsubseteq}| + |C_{\sqsubseteq}|} = \frac{2 \cdot |C_{\sqsubseteq}|}{2 \cdot |C_{\sqsubseteq}|} = 1$$

The function $sim(C, D)$ reaches its lower value when C and D have no superclasses in common, i.e. $C_{\sqsubseteq} \cap D_{\sqsubseteq} = \emptyset$:

$$sim(C, D) = \frac{2 \cdot |C_{\sqsubseteq} \cap D_{\sqsubseteq}|}{|C_{\sqsubseteq}| + |D_{\sqsubseteq}|} = \frac{2 \cdot 0}{|C_{\sqsubseteq}| + |D_{\sqsubseteq}|} = 0$$

Finally, it is true that $sim(C, D) = sim(D, C)$, since $C_{\sqsubseteq} \cap D_{\sqsubseteq} = D_{\sqsubseteq} \cap C_{\sqsubseteq}$, as demonstrated:

$$sim(C, D) = \frac{2 \cdot |C_{\sqsubseteq} \cap D_{\sqsubseteq}|}{|C_{\sqsubseteq}| + |D_{\sqsubseteq}|} = \frac{2 \cdot |D_{\sqsubseteq} \cap C_{\sqsubseteq}|}{|D_{\sqsubseteq}| + |C_{\sqsubseteq}|} = sim(D, C)$$

3. A preliminary evaluation

Once the goal of the similarity measures presented here is to help human users with the information access, one reasonable way to evaluate such measures is through their correlation with human judgement.

An experiment by Miller and Charles [15] provided some data useful for evaluating similarity measures. In this experiment, 38 students asked to rate the similarity of the meaning of 30 pairs of nouns. The similarity between the same pairs of nouns used by Miller and Charles was computed using the WordNet thesaurus [14] and the similarity measure between concepts of Equation 6 and the ones proposed by Leacock and Chodorow [12], Lin [13], Resnick [16] and Wu and Palmer [19]. The results as well as their correlation to the results in [15] are shown in Table 1.

The value of the similarities were normalized such that they are all in a scale ranging from 0 (no similarity) to 1 (synonyms).

Table 1 shows that the correlation of our measure with the experiment from Miller and Charles is very similar to the other approaches. It also shows that the results of our measure are very similar to the ones by Wu and Palmer.

Pairs of Words	Miller & Charles	Resnick	Wu & Palmer	Lin	Leacock & Chodorow	Drumond
car, automobile	0,98	0,38	1,00	1,00	0,90	1,00
gem, jewel	0,96	0,66	1,00	1,00	0,90	1,00
journey, voyage	0,96	0,36	0,93	0,69	0,72	0,93
boy, lad	0,94	0,47	0,92	0,82	0,72	0,95
coast, shore	0,93	0,56	0,92	0,97	0,72	0,92
asylum, madhouse	0,90	0,72	0,95	0,98	0,72	0,95
magician, wizard	0,88	0,74	1,00	1,00	0,90	1,00
midday, noon	0,86	0,65	1,00	1,00	0,90	1,00
furnace, stove	0,78	0,16	0,53	0,22	0,38	0,59
food, fruit	0,77	0,05	0,36	0,13	0,38	0,33
bird, cock	0,76	0,48	0,95	0,80	0,72	0,95
bird, crane	0,74	0,48	0,86	0	0,55	0,86
tool, implement	0,74	0,44	0,92	0,92	0,72	0,93
brother, monk	0,71	0,69	0,93	0,25	0,72	0,95
crane, implement	0,42	0,23	0,71	0	0,49	0,75
lad, brother	0,42	0,16	0,67	0,29	0,49	0,78
journey, car	0,29	0,00	0,13	0	0,24	0,12
monk, oracle	0,28	0,16	0,53	0,23	0,38	0,67
cemetery, woodland	0,24	0,05	0,31	0,08	0,32	0,25
food, rooster	0,22	0,05	0,24	0,10	0,24	0,24
coast, hill	0,22	0,41	0,71	0,71	0,49	0,71
forest, graveyard	0,21	0,05	0,31	0,08	0,32	0,25
shore, woodland	0,16	0,09	0,55	0,14	0,45	0,43
monk, slave	0,14	0,16	0,67	0,25	0,49	0,78
coast, forest	0,11	0,09	0,50	0,13	0,41	0,40
lad, wizard	0,11	0,16	0,67	0,27	0,49	0,78
chord, smile	0,03	0,18	0,50	0,27	0,30	0,50
glass, magician	0,03	0,16	0,46	0,13	0,38	0,35
noon, string	0,02	0,00	0,15	0	0,28	0,15
rooster, voyage	0,02	0,00	0,10	0	0,15	0,10
Correlation with Miller & Charles	1,00	0,77	0,77	0,74	0,82	0,75

Table 1 - Correlation of Drumond´s similarity measures with the Miller and Charles

4. Related Work

Most of the methods for determining the semantic similarity between entities within a single ontology use one of these approaches [6]: path distance and information content.

In the first approach the semantic similarity is assessed as a function of the distance between the terms in the hierarchical structure underlying the ontology [4]. The higher the distance between them, the less similar they are. The measure proposed by Leacock and Chodorow [12] uses this approach. The similarity between two concepts *C* and *D* is given as a function of the shortest path between *C* and *D* and the maximum depth of the hierarchy.

The measure defined in [19] also uses a path distance approach but it also considers the least common subsumer (LCS) operator [1] that computes the most specific generalization of the input concepts. The measure of Wu and Palmer [19] considers the depth of the LCS of the input concepts *C* and *D* and the depths of the input concepts.

The information content approach is founded on the Information Theory. According to the Information Theory [5], the information content of a sentence *C* is measured by the negative logarithm of the probability that sentence *C* is true, *P(C)*.

According to Resnick [16] the amount of information shared by two concepts is indicated in a taxonomy by the LCS of these concepts. Resnick also states that if the sentence *C* is a concept, *P(C)*, is the probability that a randomly chosen individual be an instance of *C*. Thus, Resnick defines the similarity between two concepts as the amount of information in the LCS of these two concepts.

The measure proposed by Lin [13] correlates the measures of Resnick and the one of Wu and Palmer [19]. This measure is a ratio between information content in the LCS of the input concepts and the information content in both concepts.

Some of the cited approaches are based on path distance which, according to Resnick [16], do not correlate well with human judgement. Other approaches use the LCS operator, whose computation is, in the worst case, exponential in the size of input descriptions [2].

The approach proposed here uses some of the ideas discussed but use neither a path distance approach nor the LCS operator.

5. Conclusions and Further Work

This work introduced a similarity analysis model for content based filtering based on the Semantic Web technologies. Such model is based on the similarity between concepts within a single ontology.

Two measures are proposed here: one for assessing the similarity between the semantic cases based models and another one for calculating the similarity between concepts.

The similarity between semantic case based models was not evaluated yet. Work on its evaluation is needed to determine is effectiveness.

The similarity between concepts was compared with other existing approaches. Results show that its correlation with the Miller and Charles experiment [15] is similar to the correlation of other approaches with the same experiment. Its correlation with the results from Miller and Charles is 0.75, while the correlation of the other considered approaches have ranged from 0.74 to 0.82. However, our approach does not make use of the LCS operator [1] nor needs a probabilistic model of the domain of application, which makes its complexity simpler. Further work on the evaluation of measures for computing the similarity between semantic case based model is still needed.

Experiments are currently being conducted to evaluate the benefits of the similarity analysis model by applying it in content based and collaborative filtering applications.

Acknowledgments

This work is supported by CNPq, an institution of the Brazilian Government for scientific and technologic development.

References

[1] BAADER, F., CALVANESE, D., MCGUINNESS, D., NARDI, D., & PATEL-SCHNEIDER, P. (2003). *The Description Logic Handbook.* Cambridge University Press.

[2] BAADER, F., KÜSTERS, R., MOLITOR, R., (1999). *Computing least common subsumers in description logics with existential restrictions.* In: Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99), 1999, pp. 96–101.

[3] BELKIN, N. J., & CROFT, W. B. (1992, Dezembro). Information filtering and information retrieval: two sides of the same coin? *Communications of ACM , 35* (12), pp. 29-38.

[4] BRIGHT, M. W., HURSON, A. R., & PAKZAD, S. (1994). Automated resolution of semantic heterogeneity in multidatabases. *ACM Transactions on Database Systems , 19*, pp. 212-253.

[5] COVER, T. M., & THOMAS, J. A. (1991). *Elements of Information Theory.* New York: Wiley.

[6] D'AMATO, C., FANIZZI, N., & ESPOSITO, F. (2005). A semantic similarity measure for expressive Description Logics. *CILC 2005, Convegno Italiano di Logica Computazionale.* Rome, Italy.

[7] DRUMOND, L., GIRARDI, R., & LEITE, A. (2007). A Case Study on the Application of the MAAEM Methodology for the Specification Modeling of Recommender Systems in the Legal Domain. *Proceedings of the 9th International Conference on Enterprise Information Systems ICEIS 2007* (pp. 155-160). Funchal: INSTICC.

[8] DRUMOND, L., GIRARDI, R., & LEITE, A. (2007). Architectural Design of a Multi-Agent Recommender System for the Legal Domain. *Proceedings of the Eleventh International Conference on ARTIFICIAL INTELLIGENCE and LAW (ICAIL 2007)* (pp. 183-188). Palo Alto, EUA: ACM Press.

[9] DRUMOND, L., GIRARDI, R. (2008). *A Multi-agent Legal Recommender System*. Journal of Artificial Intelligence and Law (to appear).

[10] GIRARDI, R. (1995). *Classification and Retrieval of Software through their Descriptions in Natural Language.* Genebra, Suíça: Imprimerie de l´Université de Geneve.

[11] GIRARDI, R., & IBRAHIM, B. (1995). Using English to Retrieve Software. *The Journal of Systems and Software, Special Issue on Software Reusability* , pp. 249-270.

[12] LEACOCK, C., & CHODOROW, M. (1998). Combining Local Context and WordNet Similarity for Word Sense Identification. In C. Fellbaum (Ed.), *WordNet: A Lexical Reference System and its Application* (pp. 265–283). Cambridge, MA: MIT Press.

[13] LIN, D. (1998). An Information-Theoretic Definition of Similarity. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 296-304). Morgan Kaufman, San Francisco, CA.

[14] MILLER, G. A. (1995). WordNet: a lexical database for English. In: *Communications of the ACM*, 38 (11), pp. 39 - 41.

[15] MILLER, G. A., & CHARLES, W. G. (1991). Contextual correlates of semantic similarity. *Language and Cognitive Processes , 6* (1), pp. 1-28.

[16] RESNICK, P. (1999). Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research* , pp. 95-130.

[17] SHADBOLT, N., HALL, W.; BERNERS-LEE, T. The Semantic Web revisited. Intelligent Systems, v. 21, p. 96-101, 2006.

[18] WANGENHEIM, G., & WANGENHEIM, A. (2003). *Raciocínio Baseado em Casos* (Vol. I). Curitiba: Manole.

[19] WU, Z., & PALMER, M. (1994). Verb semantics and lexical selection. *32nd Annual Meeting of the Association for Computational Linguistics*, (pp. 133–138).

Fuzziness in the Semantic Web: Survey and Future Directions

Seyed Koosha Golmohammadi, Marek Reformat, and Witold Pedrycz
Department of Electrical and Computer Engineering, University of Alberta, Canada
{koosha, reform, pedrycz}@ece.ualberta.ca

Abstract

The current Web includes billions of web pages and is rapidly growing. Therefore extraction of relevant information from the web is not trivial. Providing web services and improving man/machine interoperability are important issues that should be satisfied even in the presence of incomplete and inconsistent information. This paper reviews current research works on representing uncertainty and approximate reasoning in the web environment. We also examine methodologies that can address situations that involve uncertainty. We focus on fuzzy methods.

1. Introduction

World Wide Web Consortium (W3C), founded in 1994, is an international consortium working on development of Web standards and guidelines that address many critical aspects of the Web. For example, information exchange among web applications, better utilization of web technologies, as well as interaction between humans and computers. In a nutshell, W3C's mission is to make the Web as useful as possible to as many users as possible.

In 2001, the inventor of the Web and director of W3C Tim Berners-Lee introduced a new vision of the World Wide Web that is called the Semantic Web. He envisioned an environment where software agents are capable of analyzing the web contents and performing many tasks on user behalf at different levels of difficulty. The most important novelty of the Semantic Web is application of ontology as a means for effective integration and sharing of information - "people can't share knowledge if they do not speak a common language" [9].

Ontologies are widely discussed in Artificial Intelligence and have a long history in philosophy. They support knowledge sharing through well defined and partially ordered descriptions of concepts. Ontology is an "explicit specification of conceptualization" [15]. In other words, ontology is a formal description of categories (concepts), their properties (known as slots) representing various features and attributes, and restrictions imposed on the slots. The combination of ontology and a set of individuals (instances of categories) constitute a knowledge base.

The expression of concepts and their relationship in ontology relies on the assumption that all the existing knowledge components in the web are accurate. However in the real world, the contents of the web are commonly imprecise or even contradicting. The dilemma of utilizing traditional approaches in development of ontologies becomes more challenging due to the exponential growth of web contents with various degrees of uncertainty.

In this paper we review the current state of utilization of concepts of uncertainty and approximate reasoning in the Web environment. This includes methods designed for representing and reasoning with knowledge when Boolean yes/no values are inapplicable. There are different approaches applied to situations that involve uncertainty such as fuzzy sets, probability theory, belief functions, rough sets and random sets. The most commonly used approaches to deal with uncertainty in the Web are Bayesian models [11, 19] and fuzzy logic. In this paper we focus on fuzzy approaches. The objectives of this paper are:

1. to represent web utilization situations that would benefit from the application of uncertainty and approximate reasoning;
2. to review methodologies that can be applied to these situations focusing on fuzzy approaches.

An extensive study of related works has been performed to address the above objectives. We hope the result of this research will bring better understanding of the concept of uncertainty in the web environment and the need for its inclusion in development of new web technologies. We also stress that there is a need for a standard representation of uncertainty in the web environment. Currently, there are no web standards addressing the issues of representing uncertainty and reasoning with uncertainty.

The rest of this paper is structured as follows. Section 2 discusses the issue of uncertainty in the context of the Web. Section 3 reviews the technologies and definitions employed in this paper: uncertainty representation, principles of fuzzy theory, and a brief description of the Semantic Web. Section 4 examines approaches used for representing knowledge based on the concept of fuzziness. Section 5 discusses different applications of fuzzy methods to handle uncertainty in the semantic web framework, and finally Section 6 concludes the paper with a discussion.

2. Uncertainty and Web Utilization

2.1 Sources and Nature of Uncertainty

The Web is consisted of immense amount of data. Information retrieval from this extremely huge source is not immune to inconsistencies or uncertainties. Uncertainty or imprecision on the web can be related to two main factors: first, even in extremely accurate measurements we are uncertain about the implications; and second, the human perception [42] is fundamentally unable to conduct completely accurate measurements.

The Uncertainty Reasoning for the World Wide Web Incubator Group (URW3 XG) created under the W3C is dedicated to define reasoning and representation of uncertainty on the web and related technologies more appropriately. URW3 considers two facets of uncertainty:

1. aleatory: uncertainty is an inherent property of the world;
2. epistemic: uncertainty is due to someone's lack of knowledge.

In the first case we can assign degrees of truth and in the second we might assign different possibility degrees to possible alternatives. Furthermore, we can consider five types of uncertainties that may occur on the web: inconsistency, ambiguity, vagueness, randomness, and incompleteness. Examples of the above uncertainty types on the web scale are discussed in the next section.

2.2 Example Scenarios

Multiple aspects of the Web can be associated with uncertainty. The following example scenarios are just a few that represent the most intuitive illustration of needs that uncertainty can address.

- *Information correctness and availability* – it is the essence of the Web, and such issues as partially correct or even incorrect information or lack of information have to be addressed. Representation and reasoning with uncertainty provide ways for drawing conclusions and making decision in such circumstances.
- *Information precision* – information acquired by a user can be inherently imprecise. For example, weather forecast. Standards for representing and reasoning with uncertainty enable utilization of such information.
- *Concept mapping between ontologies* – the Semantic Web vision is based on ontologies and interaction among them. The issue of expressing degrees of similarity between concepts is related to vagueness and ambiguity.
- *Identification and Composition of Web Services* – any activities related to identification of services requested by a user and building a complex services based on a simple ones have to be equipped with methods and techniques that address the problem of imperfect match between user's request and available services.

3. Preliminaries

3.1 Uncertainty Representation

3.1.1 General Approach

A number of different approaches for representing uncertainty can be found in [22]. Below we shortly review descriptions of the most intuitive ones:

- *Probability theory*: Uncertainty means assigning a number between 0 and 1 to subsets of alternatives. This number – probability – represents the likelihood that the desired alternative is in a subset.
- *Fuzzy set and fuzzy measure theories*: Fuzzy sets are capable of expressing imprecision and vagueness. In fuzzy sets, our focus is not on a matter of affirmation or denial, but rather on a matter of degree. A number of special classes of measures are used: plausibility and belief measures, as well as the classical probability measures. Fuzzy measures can indicate levels of information sufficiency to determine if an element belongs to a specific set.
- *Rough set theory*: Uncertainty about an element belonging to a set is expressed in terms of two subsets, a lower approximation and an upper approximation.

Among these three approaches we are going to focus on fuzzy-based approach [8, 12, 23, 37]. There is a fundamental difference in the semantics of fuzzy logic and probabilistic logic. In fuzzy logic, a statement can be true to a certain extent or an entity belongs to a class to a certain degree. This degree is assumed to be known with certainty. In probabilistic reasoning, there

is a probability that a statement is true or false. In this case the statement itself is either true or false, but neither both nor something in between. Hence fuzzy logic sees the world as continuous instead of binary, while probabilistic logics make a claim about the randomness of the world or the observer's state of certainty [35].

3.1.2 Principles of Fuzziness

Real situations are very often not crisp and deterministic therefore they cannot be described precisely. They are very often uncertain or vague in a number of ways. One aspect of uncertainty is related to lack of information about the future state of the system. This type of uncertainty is handled appropriately by probability theory and statistics. It is assumed that the events are well defined. This is in contrast to the vagueness concerning the description of the semantic meaning of the event, phenomena or statements, which is called fuzziness [28, 39, 43].

Fuzziness can be found in many areas of daily life. It is particularly frequent, however, in all areas in which human judgment, evaluation, and decisions are important. One of the most important reasons for that is that human daily communication uses natural languages and a good part of human thinking is done with it. In these natural languages the meaning of the words is very often vague. The meaning of a word might even be well defined, but when using the word as a label for a set, the boundaries within which objects belong to the set or not, become fuzzy or vague. Examples are words such as "birds" (how about penguins, bats, etc.?), "red flowers", but also terms such as "tall men", "creditworthy customer". In this context, two kinds of fuzziness with respect to their origins can be distinguished: intrinsic fuzziness and informational fuzziness. The former is illustrated by "tall men". This term is fuzzy because the meaning of tall is fuzzy and dependent on the context. An example of the latter is the term "creditworthy customer": a creditworthy customer can possibly be described completely and crisply if a large number of descriptors are used. However, this is more than a human being could handle simultaneously. Therefore the term, which in psychology is called a "subjective category" becomes fuzzy.

The idea of fuzzy theory was first introduced by Lotfi Zadeh at the University of California at Berkeley in the 1960s [40]. Zadeh was working on the problem of computer understanding of natural language. Natural language is not easily translated into the absolute terms of "true" and "false". Fuzzy logic includes "true" and "false" as extreme cases of truth about phenomena or statement. Fuzzy logic also includes the various states of truth in between. For example, the result of a comparison between two things could be not "tall" or "short" but "0.38 of tallness".

3.2 Semantic Web and Ontology

The concept of the Semantic Web was introduced in May 2001 in Scientific American by Tim Berners-Lee, James Hendler, and Ora Lassila [5]. Over the last years the Semantic Web has been described in many ways: an extension of the current web in which information is given well-defined meaning, a place where machines can analyze all the data on the Web [5]. A common element of all of these definitions is a reference to a new method of representing data. The formation of the Semantic Web has been led by advances in the area of data and knowledge representation.

In a nutshell, the Semantic Web can be seen as a new representation of resources on the World Wide Web. It is virtually a hub of linked information that can be accessible and operable by programs. These programs can be in a form of software agents or any other applications which are capable of handing the semantics of the information.

The new representation of resources on the web is based on usage of ontology. Ontology is a formal, explicit specification of a shared conceptualization [16]. It is a set of well-defined classes to describe data models in the specific domain. Ontology has ability to present interrelated resources. Together with their instances, ontologies work as knowledge characters to express the individual facts [30].

In the Semantic Web environment ontology is specified using Resource Description Framework (RDF). RDF is a foundation for processing metadata [31]. RDF is a standard for describing resources and information on the web. It provides interoperability between applications that exchange machine-understandable information on the Web. Resource Description Framework Schema (RDFS) is used as an ontology language supporting exchange of knowledge over the web. RDF and RDFS serve as the basic methodology of expressing web resources in the form of triples: a subject, a predicate (i.e. verb), and an object (consider them as start, label and end of the edge respectively in a labeled, directed graph). Another ontology specification language is a combination of DARPA Agent Markup Language (DAML) and Ontology Inference Layer (OIL) called DAML+OIL. It enables the creation of ontologies for any domain and the instantiation of these ontologies in the description of specific web sites. DAML+OIL enhances and extends RDFS with richer modeling

primitives [36] to represent the semantics of resources and information.

The latest web resource ontology language is Web Ontology Language (OWL), which has been proposed as the recommendation by W3C. OWL has many correspondences with Description Logic (DL). DLs [3] are considered the most important formalism to represent knowledge of an application domain. They combine traditions of Frame-based systems, Semantic Networks and KL-ONE-like languages, Object-Oriented representations, Semantic data models, and Type systems. OWL is not only for representing information on the web, but it also improves the capability to process the information and increases the interoperability among software agents [24]. OWL defines a family of three languages: OWL Lite, OWL Full, and OWL-DL.

4. Fuzziness and Ontology Languages

The developments related to Semantic Web, and especially the application of ontology to knowledge representation, have created a suitable setting for representing uncertainty.

Fuzzy OWL, developed in the National Technical University of Athens [32], has been proposed in 2006. In this approach, a class is defined by a membership function that returns the membership value between [0,1] representing a degree of belonging of a given object to the class. Fuzzy OWL uses crisp OWL's syntax for class, property axioms and definitions. Reasoning is done using a reasoning platform – Fuzzy Reasoning Engine (FiRE), and FiRE uses RACER DL[1] engine syntax. Fuzzy of OWL (FOWL) [29] is another extention to the OWL by fuzzy logic to capture uncertain and imprecise knowledge with modifying operators.

DL in the web ontology language (OWL-DL) corresponds to SHOIN(D)[2] description logic. In other words OWL-DL is using SHOIN description logic to represent knowledge and reason about it. Straccia presented a fuzzy extension of SHOIN(D) showing that its representation and reasoning capabilities go beyond classical SHOIN(D) [33]. A main feature of fuzzy SHOIN(D) is that the subsumption relation between classes and the entailment relation is no more a crisp yes/no problem, but it becomes now fuzzy, i.e. is established to some degree. Since many languages such as OWL-DL are based on DLs therefore a better understanding of DLs is indispensable for Semantic Web researchers [20].

In addition, research is being conducted in the area of introducing rules to OWL. Semantic Web Rule Language (SWRL) is a proposal that combines OWL (DL and Lite) with the Rule Markup Language (RuleML). Fuzzy-SWRL (f-SWRL) is a fuzzy extension of Semantic Web Rule Language [26]. In both the antecedent and consequent of f-SWRL rules atoms can have weights between [0,1]. f-SWRL provides a powerful and flexible knowledge representation and very convenient for multimedia domain.

The results of work on fuzzy ontologies are reported in [34]. A framework called Fuzzy Ontology Generation frAmefork (FOGA) has been developed. It combines fuzzy logic and Formal Concept Analysis (FCA) [13] to represent the uncertainty information by a value in the range from 0 to 1 (linguistic variables are no longer needed). FOGA automatically generates fuzzy ontologies based on data with uncertainty.

5. Fuzziness in the Semantic Web Systems

First steps in introduction of fuzziness to knowledge representation are associated with first applications of fuzziness to building web applications.

A collaborative filtering multi-agent model was introduced in [17]. It relies on fuzzy linguistic approach [41]. The retrieval capabilities of this model do not utilize a user's profile what is seen as a drawback. This limitation is addressed in the further work [18] through modifying the model by incorporating a user profile to improve information retrieval. The new model combines semantic web technologies with a dynamic user's profile relying on fuzzy linguistic techniques.

Haibin and Yan proposed a framework called soft Semantic Web Services agent (soft SWS agent) [38] providing high quality semantic web services using fuzzy neural networks and genetic algorithms. The core of soft SWS agent is the Intelligent Inference Engine (IIE) that uses a four-layer fuzzy neural network. Linguistic variables entered to the network are transformed into output variables after undergoing fuzzy processing.

A concept-matching information retrieval system – a system that retrieves web pages that are conceptually related to the implicit concepts of the query – is introduced in [14]. The system uses fuzzy synonymy and fuzzy generality interrelations as a means of representing word interrelations. It applies Synonymy-Based Concept Representation Model (FIS-CRM) to extract the concepts from web pages and user's

[1] www.sts.tu-harburg.de/~r.f.moeller/racer

[2] SHOIN(D) forms the core of OWL-DL (OWL-DL is a syntactic variant of SHOIN(D))

queries. The vectors used in FIS-CRM are fuzzy values representing occurrences of concepts instead of terms.

Acampora and Loia describe a multilayer architecture to design Ambient Intelligent (AmI) [4] systems providing efficient and uniform utilization of control activities [1]. This multiplayer architecture employs markup-based technologies to transform rough information on sensors, actuators and services towards "smart data". In particular they are using Fuzzy Markup Language (FML) [2] to provide fuzzy web services. FML language is a novel computer language used to model control systems based on fuzzy logic theories. The main feature of FML is the transparency property: the FML programs can be executed on different hardware without additional efforts. This property is fundamental in ubiquitous computing environment where computers are available throughout the physical environment and appear invisible and transparent to the user.

Nikravesh introduces a new architecture for semantic web search engines based on Fuzzy Conceptual Model (FCM) to handle the ambiguity and imprecision of the *concept* on the Internet [25]. In the FCM approach, the *concept* is defined by a series of keywords with different weights depending on the importance of each keyword. Ambiguity of *concepts* is defined by a set of imprecise concepts described using fuzzy concepts. The fuzzy concepts are related to a set of imprecise words identified by context. Imprecise words can be translated into precise words using ontology and ambiguity resolution through clarification dialog.

A popular statement about the Web – "anyone can say anything about anything" means that information can be of different trustworthiness. The agents in the semantic web framework have to be able to make judgments to choose a single, most reliable source from alternative sources of information. Trust is an essential component of the semantic web vision [5-7].

In [10], the authors treat trust as a degree that a source can be trusted. They introduce a model that takes into account partial trust, distrust and ignorance simultaneously. This model is particularly useful when the trustworthiness of many sources of information is unknown for a user at the beginning. This does not mean the user distrust all sources but eventually further evidence reveals their credibility.

6. Discussion

6.1 Knowledge Representation

Currently, typical ontology formalisms have very limited or no capabilities to represent different aspects of uncertainty. Uncertainty is inherently present in many application domains. This has initiated research activities leading to additions of elements of probabilistic and fuzzy theories to existing knowledge representation formats. As we presented above, fuzziness has already been introduced to ontologies. Some of the items that sill need attention are:

- inclusion of fuzziness to DL and OWL-DL and development of reasoning systems taking full advantaged of introduced fuzziness; this would also include fuzzy rules;
- construction of fuzzy ontologies where relationships (is-a, as well as relationships defined by object properties) among concepts are expressed by a number in the range from 0 to 1 [34], development of methods for automatic construction of such ontolgies and their interaction with normal (crisp) ontologies;
- development of fuzzy-based methods and algorithms for matching and comparison of ontologies.

It should be also stated, that fuzzy logic cannot address all faces of imperfect knowledge. For example, rough sets theory [27] has been proposed to deal with indiscernibility of objects. Therefore, fuzzy methods used to represent ontologies can be combined with rough sets to handle uncertainty in DLs [21].

6.2 Web Services

The Semantic Web promises a change in a way a human will use the Internet. According to its motto web agents should be able to act on behalf of users and like users. It seems that fulfillment of that promise means existence of agents that have capabilities to deal with uncertainty. It is essential to develop agents that can use imprecise information and reason about it. In particular, the following issues should be addressed:

- selection of most suitable services in the presence of partial information;
- integration of atomic services when they are not fully compatible;
- supporting user in human-centric (like) multi-criteria decision making when multiple alternatives and service providers are available.

With the assumption that information uncertainty can be expressed by ontology, there is a need for methods and techniques able to automatically identify levels of information uncertainty, store that information, and reason about it. Utilization of all those things depends on existence of open-source and commercial reasoning engines capable of handling uncertainty.

References

[1] G. Acampora, V. Loia, Enhancing the FML vision for the design of open ambient intelligence environment. *Systems, Man and Cybernetics, 2005 IEEE International Conference,* vol. 3, pp. 2578 - 83, 2005.

[2] G Acampora, V. Loia, Fuzzy Control Interoperability and Scalability for Adaptive Domotic Framework, *IEEE Transactions on Industrial Informatics,* vol. 1, pp. 97-111, 2005.

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook,* Cambridge University Press, 2002.

[4] T. Basten, M. Geilen, H. de Groot, Ambient Intelligence: Impact on Embedded System Design, Kluwer Academic Publishers, 2003.

[5] T. Berners-Lee, J. Hendler, and O. Lassila, *The Semantic Web. Scientific American,* vol. 284, pp. 34-43, 2001.

[6] T. Berners-Lee, *Weaving the Web, Harper*, 1999.

[7] T. Berners-Lee, W. Hall, J. Hendler,K. O'Hara, N. Shadbolt, D.Weitzner, A framework for web science, *Foundations and Trends in Web Science*, pp. 1-130, vol. 1, 2006.

[8] C. V. Damasio, L. M. Pereira, Antitonic logic programs, *6th International Conference on Logic Programming and Nonmonotonic Reasoning,* pp. 748-59, 2001.

[9] T. H. Davenport and L. Prusak, *Working Knowledge: How Organizations Manage What They Know*, Cambridge, MA: Harvard Business School Press, 1998.

[10] M. De Cock and P. P. da Silva, A Many Valued Representation and Propagation of Trust and Distrus. *Lecture Notes in Computer Science*, vol. 3849. pp. 114-20, 2006.

[11] Z. Ding et al. BayesOWL: Uncertainty modeling in Semantic Web ontologies. Z. Ma, editor, *Soft Computing in Ontologies and Semantic Web, Studies in Fuzziness and Soft Computing*, vol. 204, pp. 3-29, 2006.

[12] R. Ebrahim, Fuzzy logic programming. *Fuzzy Sets and Systems*, vol. 117, pp. 215-30, 2001.

[13] B. Ganter and R. Wille, *Formal Concept Analysis: Mathematical Foundations*. Springer, 1999.

[14] P. J Garces, J A Olivas, and F. P. Romero, Concept-Matching IR System Versus Word-Matching Information Retrieval Systems: Considering Fuzzy Interrelations for Indexing Web Pages, *Journal of the American Society for Information Science and Technology,* vol. 57, pp. 564-76, 2006

[15] T. R. Gruber, Toward Principles for the design of Ontologies Used for Knowledge Sharing, *International Journal of Human and Computer Studies*, pp. 907-28, 1995.

[16] T. R. Gruber, A translation approach to portable ontology specifications, *Knowledge Acquisition*, vol. 5, pp. 199-220, 1993.

[17] E. Herrera-Viedma, C. Porcel, A. G. Lopez, M. D. Olvera and K. Anaya, A Fuzzy Linguistic Multi-agent Model for Information Gathering on the Web Based on Collaborative Filtering Techniques. *Lecture Notes in Computer Science*, pp. 3-12, 2004.

[18] E. Herrera-Viedma, E. Peis, J. M. Morales-del-Castillo and K. Anaya, Web-based Service Information Systems based on Fuzzy Linguistic Techniques and Semantic Web Technologies, *Studies in Computational Intelligence (SCI)*, vol. 37, pp. 647–66, 2007.

[19] M. Holi and E. Hyvonen, Modeling Uncertainty in Semantic Web Taxonomies, *Studies in Fuzziness,* vol. 204, pp 31-46, 2006.

[20] I. Horrocks, P. F. Patel-Schneider, F. van Harmelen, From SHIQ and RDF to OWL: the making of a Web Ontology Language, *Journal of Web Semantics,* vol. 1, pp. 7-26, 2003.

[21] P. Klinov, L. J. Mazlack, Fuzzy Rough Approach To Handling Imprecision in Semantic Web Ontologies, *Fuzzy Information Processing Society (NAFIPS 2006) Annual Meeting of the North American,* pp. 142-7, 2006.

[22] G. J. Klir, The Many Faces of Uncertainty, In: B.M Ayubb and M.M. Gupta, editors, Uncertainty Modeling and Analysis: Theory and Applications, *Elsevier Science*, pp. 3-19, 1994.

[23] C. Mateis, Extending disjunctive logic programming by t-norms. In LPNMR '99: Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning, pp. 290-304, 1999.

[24] D.L. McGuinness, F. van Harmelen (eds.), OWL Web Ontology Language Overview. W3C Recommendation. 10 February 2004, Available at http://www.w3.org/TR/2004/REC-owl-features-20040210/. Accessed 03/12/2008.

[25] M. Nikravesh. Beyond the Semantic Web: Fuzzy Logic-Based Web Intelligence, Zongmin M (ed) *Soft Computing in Ontologies and Semantic Web, Studies in Fuzzyness and Soft Computing,* vol. 204, pp. 149-242, 2006.

[26] J. Z. Pan, G. Stamou, V. Tzouvaras, and I. Horrocks, f- SWRL: A Fuzzy Extension of SWRL, *Journal on Data Semantics VI,* pp. 28-46, 2006.

[27] Z. Pawlak, Rough sets, *International Journal of Computer and Information Sciences*, vol. 11, pp. 341-56, 1982.

[28] W. Pedrycz, F. Gomide, *An Introduction to Fuzzy Sets: Analysis and Design,* MIT Press, 1998.

[29] Y. Quing, W. Jinlin, Extending Ontology Language for Semantic Web, *Computational Intelligence and Security Workshops (CISW 2007),* pp. 116-119, 2007.

[30] F. Scott, W. D. Lewis, D. T. Langendoen, An Ontology for Linguistic Annotation 1-2. *Fourteenth Innovative Applications of AI Conference,* pp. 11-9, 2002.

[31] S. Staab, M. Erdmann, A. Maedche, S. Decker, An Extensible Approach for Modeling Ontologies in RDF(S) *ECDL 200 Workshop on the Semantic Web*, pp 11-22, 2000.

[32] G. Stoilos, N. Simou, G. Stamou, S. Kollias, Uncertainty and the Semantic Web, *IEEE Intelligent Systems,*vol. 21, pp. 84-7, 2006.

[33] U. Straccia. Towards a Fuzzy Description Logic for the Semantic Web, *2nd European Semantic Web Conference (ESWC-05),* pp. 167-81, 2005.

[34] Q. T. Tho, S. C. Hui, A.C.M. Fong and T. H. Cao. Automatic fuzzy ontology generation for semantic Web, *IEEE Transactions on Knowledge and Data Engineering,* vol. 18, pp. 842-56, 2006.

[35] C. J. Thomas and A. Sheth, *On the Expressiveness of the Languages for the Semantic Web – Making a Case for 'A Little More'*, in E. Sanchez (Editor), "Fuzzy Logic and the Semantic Web", Elsevier, 2006.

[36] F. van Harmelen, P. F. Patel-Schneider I. Horrocks, *Reference description of the DAML+OIL ontology markup language, 2001* http://www.daml.org/2001/03/reference.

[37] P. Vojtas, Fuzzy logic programming. *Fuzzy Sets and Systems,* vol. 124, pp. 361-70, 2001.

[38] H. Wang, Y-Q Zhang, R. Sunderraman, Soft Semantic Web Services Agent, *Fuzzy Information, 2004. Processing NAFIPS '04. IEEE Annual Meeting of the North American,* vol. 1, pp. 126-9, 2004.

[39] L. A. Zadeh, A Fuzzy-Set Theoretic Interpretation of Linguistic Hedges. *Journal of Cybernetic*, vol. 2, pp. 2-34, 1972.

[40] L. A. Zadeh, Fuzzy Sets. *Information and Control*, vol. 8, pp. 338-53, 1965.

[41] L. A. Zadeh, The concept of a linguistic variable and its applications to approximate reasoning. Part I, In *Information Sciences vol.* 8, pp. 199-249, 1975. Part II, *Information Sciences vol.* 8, pp. 301-357, 1975. Part III, *Information Sciences vol.* 9, pp. 43-80, 1975.

[42] L. A. Zadeh, Toward a perception-based theory of probabilistic reasoning with imprecise probabilities, *Journal of Statistical Planning and Inference,* vol. 105, pp. 233-64, 2002.

[43] H. J. Zimmermann, Fuzzy Set Theory - and Inference Mechanism. *Mathematical Models for Decision Support,* pp. 727-741, 1988.

A Language-based Approach to Addressing Reliability in Composite Web Services

Onyeka Ezenwoye
Electrical Engineering and
Computer Science Department
South Dakota State University
Brookings, SD 57007
Email: *onyeka.ezenwoye@sdstate.edu*

S. Masoud Sadjadi
School of Computing and
Information Sciences
Florida International University
11200 SW 8th Street, Miami, FL 33199
Email: *sadjadi@cs.fiu.edu*

Abstract

With Web services, distributed applications can be encapsulated as self-contained, discoverable software components that can be integrated to create other applications. BPEL allows for the composition of existing Web services to create new higher-function Web services. We identified that the techniques currently applied at development time are not sufficient for ensuring the reliability of composite Web services In this paper, we present a language-based approach to transparently *adapting BPEL processes to improve reliability. This approach addresses reliability at the Business process layer (i.e the language layer) using a code generator, which weaves fault-tolerant code to the original code and an external proxy. The generated code uses standard BPEL constructs, and therefore, does not require any changes to the BPEL engine.*

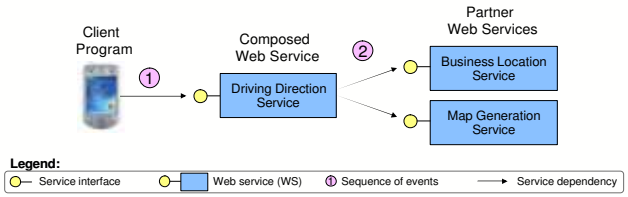
Keywords: Web service composition, Reliability, Adaptability, Business Process.

1 Introduction

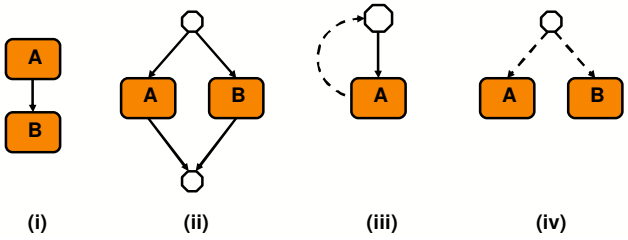
Web services are gaining acceptance as the predominant standards-based approach to building open distributed systems. With Web services, distributed applications can be encapsulated as self-contained, discoverable and Internet-accessible software components that can be integrated to create other applications. The fundamental aspects of Web services can be summarized as follows: (1) strict separation of service interface description, implementation, and binding; (2) declarative policies and Service Level Agreements (SLAs) to govern service interactions; and (3) loosely coupled, standards-based and message-centric interactions between autonomous and replaceable service components [1]. To facilitate flexibility and interoperability, Web services are described using a standard, machine-readable, XML-based language called *Web Service Description Language*. This service description provides the details necessary to interact with the service, including message formats that detail the operations, transport protocols, and location. Finally, interaction with Web services is achieved through SOAP messaging.

The family of specifications that make up the Web service standards includes a specification for service composition known as *Business Process Execution Language* (BPEL). BPEL allows for the composition of existing Web services to create new higher-function Web services [2]. BPEL is used to define workflows that represent composite services. The composite services, also known as *business processes*, contain activities that coordinate the interaction between the partner services in the composition. Figure 1(a) illustrates a business process that is a composition of two service: (1) a service that retrieves the addresses of nearby businesses; and (2) a service that gets the driving directions to a given address. Figure 1(b) depicts a basic set of workflow patterns that are supported by BPEL. In the sequence pattern (Figure 1(b)(i)), an activity in a process is enabled after the completion of another activity in the same process. Parallelism (Figure 1(b)(ii)), allows activities to be executed simultaneously. Loops (Figure 1(b)(iii)), allow for one or more activities to be executed repeatedly. In the choice pattern (Figure 1(b)(iv)), a number of branches are chosen and executed as parallel threads. Based on these basic patterns, more sophisticated constructs can be built [3].

As the use of Web services continues to grow, so has the need to deliver reliable service compositions with precise Quality of Service (QoS) attributes covering functional correctness, performance and dependability [1]. This is because current Web services standards provide limited constructs for specifying exceptional behavior and recovery



(a) A Business Process that integrates remote components to create a new composite component for providing driving directions.



(b) Selected workflow patterns supported by BPEL: (i) Sequence, (ii) Parallelism, (iii) Loop, (iv) Choice.

Figure 1.

actions. Currently, BPEL is a composition language that mainly concentrates on modeling the business process in terms of interacting Web services but does not consider the behavior of such models at runtime.

While it is relatively easy to make an individual service fault-tolerant, addressing reliability and availability of Web services collaborating in multiple application scenarios is a challenging task. This is because the integration of multiple services, which are potentially developed and maintained on autonomous heterogeneous environments, introduces new levels of complexity in management. Thus the composed service has no influence over the factors affecting QoS provision and partner services can spontaneously appear and disappear over on the Internet. Moreover, services may fail because of problems in their execution such as network faults, overload and lack of resources [1].

Given the unreliability of communication channels, the unbounded communication delays, and the autonomy of the interacting services, it is difficult for developers of composite services to anticipate and account for all the dynamics of such interactions. There is therefore a need for adaptability in composed services to make them more robust and dependable. The need for adaptability is particularly evident in complex long-running applications as is found in scientific Grid computing. In Grid computing, computational and storage resources are exposed as an extensible set of networked services that can be aggregated to create higher-function applications [4]. These highly available applications need to remain operational and rapidly responsive even when failures disrupt some of the nodes in the system.

In this paper, we present a *systematic* approach to making existing aggregate Web services more tolerant to the failure. We demonstrate how a composite Web service, defined as a BPEL process, can be instrumented automatically to monitor its partner Web services at runtime. To achieve this, events such as faults and timeouts are monitored from within the adapted process. We show how our adapted process is augmented with a proxy that dynamically replaces failed services. In doing this, we improve the fault tolerance and performance of BPEL processes by transparently adapting their behavior. By *transparent*, we mean the following; first, the adaptation preserves the original behavior of the business process and does not tangle the code that provides self-healing and self-optimization behavior with that of the business process; and second, the fault-tolerant approach does not need any modification of the BPEL engine[1]. This transparency is achieved by using a *dynamic* proxy that encapsulates the autonomic behavior (adaptive code).

The rest of this paper is is structured as follows. Section 2 provides a background in addressing reliability in composite components. Section 3 overviews our approach and gives a brief introduction to the RobustBPEL framework, we also describes the dynamic proxy. Section 4 contains some related work. Finally, some concluding remarks are provided in Section 5.

2 Addressing Reliability in Composite Components

The goal of *fault-tolerance* is to improve *dependability* in a *system* by enabling it to perform its intended functions in the presence of a given number of faults [5]. There exists several definitions of dependability. These definitions often depend on the attributes (e.g., availability, reliability and safety) of the system that are being defined as a criterion to decide whether or not a system is dependable at a given time. The attribute defined may depend on the intended use of the system [6].

In general, dependability is based on the notion of *reliance* in the context of interacting components. It associates to the relation *depends upon*, where a component *A* depends upon a component *B* if the correctness of *B's* service delivery is necessary for the correctness of *A's* service delivery [6]. This relationship is typical of composite services since they are entirely dependent on interaction with partner services. An error may propagate from a partner to the composite thereby creating new errors.

[1] A BPEL engine is a virtual machine that executes BPEL grammar

Our work focuses on the *reliability* attribute of dependability with a specialization on *robustness* as a *secondary attribute*. Avizienis [6] defines reliability as the continuity of *correct* service, it defines robustness as dependability with respect to *external* faults. Techniques for achieving dependability that are applied at development time are not sufficient enough for ensuring the reliability of composite Web services that are expected to dynamically discover and assemble components, configure themselves, and operate securely and reliably in a completely automated manner. This calls for the development of new reliability techniques that introduce *autonomic* functionality to address these challenges.

New reliability techniques for service compositions can be developed at four layers. Figure 2 shows the different layers at which reliability techniques can be applied.

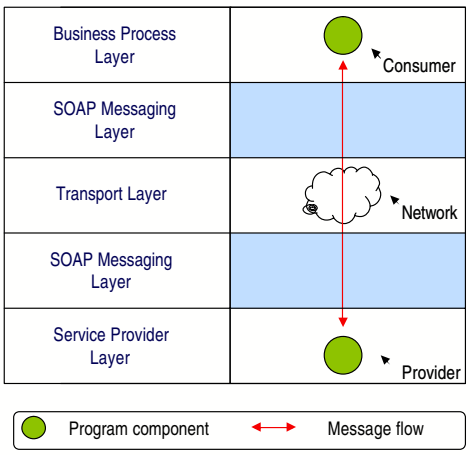


Figure 2. Layers to apply reliability techniques

Service provider layer: At this level, reliability focuses on the service hosting environment. Here, reliability can be achieved by techniques that provide redundancy of computation and data, load sharing to improve performance and fault tolerance, and clustering which interconnects multiple servers to avoid single point of failure [1].

Transport layer: At this level, the focus is on implementing reliable messaging for Web services at the transport layer. Since the reliability of SOAP messaging is dependent on the underlying transport layer, techniques in this layer center on using message-oriented middleware (MOM) [7] to ensure reliability and robustness of message traffic.

SOAP messaging layer: Addressing reliability at this layer focuses on extending SOAP messages to include reliability properties that allow messages to be delivered reliably between services in the presence of component, system, or network failures.

Business process layer: Reliability at this layer aims to provide dependable composition of Web services through advanced failure handling and compensation-based transaction protocols [1]. Efforts in this layer can be categorized into two groups; *language-based* and *non language-based* approaches. *Language-based* techniques provide advanced failure handling and adaptability by augmenting the process logic with additional language constructs while *non-language* based approaches focus specifically on the process supporting infrastructure such as the execution engine.

Our work fits into this category by enabling adaptability in BPEL process to address the concerns raised above. One might argue that BPEL should be extended with constructs to handle those concerns. However, this would increase the complexity of the language and it is also against the principle of separation of concerns. Constructs for specifying exceptional behavior and recovery actions should be modularized and externalized and not scattered and tangled with the service implementation. Entangling the logic for exceptional behavior and recovery actions with the business logic of the application negatively impacts maintainability and adaptability.

3 Overview of Our Approach

We developed RobustBPEL [8] as part of the transparent shaping programming model. Using RobustBPEL, we can automatically generate an *adapt-ready* version of an existing BPEL process. In a typical composed Web service (see Figure 1(a)), a request is first sent by the client program, then the composite Web service interacts with its partner Web services and responds to the client. If one of the partner services fails, then the whole process is subject to failure. To avoid such situations, adapt-ready version of the original composed service monitors the behavior of it partners and tries to tolerate their failure. As monitoring all the partner Web services might not be necessary, the developer can select only a subset of Web service partners to be monitored. The adapt-ready process monitors selected Web services and in the presence of faults it will forward the corresponding request to a *proxy*. The proxy is generated specifically for this adapt-ready process and provides the same interface as those of the monitored Web services. The proxy in its turn forwards the request to a *substitute* Web service.

In this work, we make the following assumptions: (1) two services are *substitute*, if they implement the same interface; (2) Web service partners are *stateless* and *idempotent*. It is possible for two applications to be functionally equivalent without necessarily having the exact same interface. When this occurs, a wrapper interface/service can be used to harmonize the differences in their interfaces.

Given the rapid uptake of the service oriented program-

ming model, we expect the emergence of numerous services that are functionally equivalent and thus can be substituted. For instance, in our driving-direction example (Figure 1(a)), if the default map generation service provided by Google fails, it should be possible to substitute this service with that of MSN, Yahoo! or Mapquest. Also, in Grid programming environments where scientific applications are run on computational Grids, a failed (or slow) Grid service can be replaced by another service on the Grid. Thus, in our approach, we associate an adapt-ready composed service with a *dynamic proxy* (which is also a Web service) and its job is to discover and bind to *substitute* Web services.

3.1 High-Level Architecture

Figure 3 illustrates the architectural diagram of an application using an adapt-ready BPEL process augmented with its corresponding dynamic proxy. This figure shows the steps of interactions among the components of a typical adapt-ready BPEL process. Similar to a static proxy, the interface for the generated dynamic proxy is exactly the same as that of the monitored Web service. Thus, the operations and input/output variables of the proxy are the same as that of the monitored invocation. When more than one service is monitored within a BPEL process, the interface for the specific proxy is an aggregation of all the interfaces of the monitored Web services. For example, the dynamic proxy in Figure 3 has pt_i and pt_j, which are the port types of the two monitored Web services (namely, WS_i and WS_j). At runtime, if a monitored service fails (or an invocation timeout occurs), the input message for that service is used as input message for the proxy. The proxy invokes the equivalent service with that same input message. A reply from the substitute service is sent back to the adapted BPEL process via the proxy.

Although the adapt-ready BPEL process remains a functional Web service and the proxy is an autonomic Web service (encapsulates autonomic attributes), functional Web services can behave in an autonomic manner by virtue of their interaction with autonomic Web services. By replacing failed and delayed services with substitutes, the proxy service provides self-healing and self-optimization behavior to the BPEL process, thereby making the BPEL process autonomic.

3.2 Incorporating Generic Hooks inside the Adapt-Ready BPEL Processes

Following the Transparent Shaping programming model [9], we first need to incorporate some generic hooks at sensitive *joinpoints* in the original BPEL process. These joinpoints are certain points in the execution path of the program at which adaptive code can be introduced at run

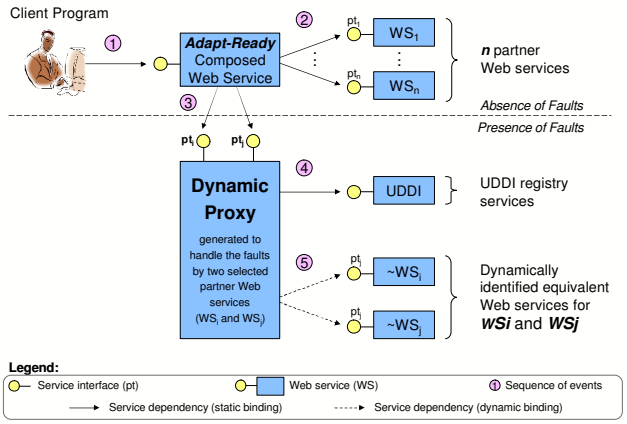


Figure 3. Architectural diagram showing the sequence of interactions among the components in an adapt-ready BPEL process augmented with its corresponding *dynamic* proxy.

time. Key to identifying joinpoints is knowing where in the BPEL process *sensing* and *actuating* are required and inserting appropriate code (hooks) to do so. Because a BPEL process is an aggregation of services, the most appropriate place to insert interception hooks is at the interaction joinpoints (*i.e.,* the *invocation* instructions). The monitoring code we insert is in the form of standard BPEL constructs to ensure the portability of the modified process.

We adapt the BPEL process by identifying points in the process at which external Web services are invoked and then wrapping each of those invocations with a BPEL `scope` that contains the desired fault and event handlers. A fault can be a programmatic error generated by a Web service partner of the BPEL process or unexpected errors from the Web service infrastructure. The unmonitored invocation is first wrapped in a `scope` container which contains fault and event handlers. The fault handlers detect any faults generated as a result of the invocation of the partner Web service. A fault-handling activity is defined, which basically forwards the request to the dynamic proxy. When a fault is generated by the partner service invocation, this fault is caught and the proxy service is invoked to substitute for the unavailable or failed service.

For the event handler, an alarm clause is used to specify a timeout. A timeout can be used, for instance, to limit the amount of time that a process can wait for a reply from an invoked Web service. If the partner service fails to reply within the time stipulated in the timeout event, a generated fault forces the monitored invocation to terminate and the proxy service is invoked as a substitute.

4 Related Work

Since Web services technology is still emerging, most of the work that aim to address the requirements for reliable and fault tolerant Web services execution are still in their infancy. These efforts can be distinguished by their focus on different layers (see Figure 2) of the Web services infrastructure . We note that our work is focused on the business process layer and as a result the work in the other layers are complementary to ours.

4.1 SOAP Messaging Layer

Some works aim to address the reliability of Web services from the *SOAP messaging layer* by addressing the issues concerning reliable transport-independent messaging. To this end, SOAP-based protocols like WS-ReliableMessaging [10] and WS-Reliability [11] strive to standardize message delivery by specifying rules for acknowledgment, message correlation, ordered delivery and so on. Such a protocol does however contribute to inefficiency if the underlying transport layer does use protocols that address reliable message delivery [1].

4.2 Transport Layer

Other approaches and technologies focus on implementing reliable massaging for Web services at the transport layer. The reliability of SOAP messaging largely depends on the underlying transport chosen. Since SOAP-over-HTTP is not reliable, attempts are being made to build messaging middleware that accept messages from sending processes and delivers them reliably to receiving processes. Reliable messaging implementations communicate across a network on behalf of senders and receivers, and have built-in transactional support to manage message conversations in the context of a larger business process [1]. Examples of message-oriented middleware are IBM WebsphereMQ [12] and Microsoft Message Queuing (MSMQ) [13]. These implementations support their own proprietary messaging APIs and protocols, as well as the standard Java Message Service (JMS) API [14]. These approaches however do not guarantee reliability for multi-hop messaging over different protocols as they assume that reliable transport protocols will be available for the entire path of the message [1, 15].

4.3 Service Provider Layer

At this layer, approaches focus on the service hosting container. Here, approaches aim to achieve reliability by using techniques that provide redundancy of computation and data, load sharing to improve performance and fault tolerance, and clustering to avoid single point of failure [1].

Dialani et al. [16] provide an approach to enabling fault tolerance in *stateful* Web services by requiring the developer to implement an interface for rollback and checkpoint.

Birman et al. [17] propose extensions to the Web services architecture to support mission-critical applications. They propose some extensions to track the health of individual Web service.

4.4 Business Process Layer

We further categorize works that focus on the business process layer into two groups: *language-based* and *non language-based*.

Non-language based approaches focus specifically on the process supporting infrastructure such as the execution engine. They include wsBus [18], which is a lightweight service-oriented middleware for transparently enacting recovery action in service-based processes. This approach is modular and separates the business logic of the process from the QoS requirements; however, this approach requires the installation of additional middleware.

Charfi et al. [19] use an aspect-based container to provide middleware support for BPEL. The process container is the runtime environment for the BPEL process. All interactions go through the container which plugs in support for non-functional requirements. This framework is different from ours because it requires a purpose built BPEL engine.

Language-based techniques provide advanced failure handling and adaptability by augmenting the process logic with additional language constructs. These approaches include BPEL for Java (BPELJ), which combines the capabilities of BPEL and the Java programming language. This combination is achieved by extending the BPEL to allow for sections of Java code to be included in BPEL process definitions. BPELJ, however, requires an extended BPEL engine that understands the additional constructs. Also, exception handling logic in BPELJ often gets tangled with the process logic, thus hampering maintainability.

Other language-based techniques include the work done by Baresi et al. [20]. In their approach, BPEL processes are monitored at run-time to check whether individual services comply with their contracts. Monitors are automatically defined as additional services and linked to the service composition via annotations in the composition. This approach achieves the desired separation of concern, however, it requires manually modifying the original BPEL process and the monitoring code is entangled with the process logic. The manual modification of BPEL code is not only difficult and error prone, but also hinders maintainability.

5 Conclusion

Techniques that are applied at development time are not sufficient enough for ensuring the reliability of composite Web services that are expected to dynamically discover and assemble components, configure themselves, and oper-

ate securely and reliably in a completely automated manner. This calls for the development of new reliability techniques that introduce *autonomic* functionality to address these challenges. New reliability techniques for service compositions can be developed at four layers, namely; (1) Service provider, (2) SOAP messaging, (3) Transport and (4) Business process layers. We presented a language-based approach to transparently adapting BPEL processes to improve reliability. This approach addresses reliability at the Business process layer.

References

[1] A. Erradi, P. Maheshwari, and V. Tosic, "A policy-based middleware for enhancing web services reliability using recovery policies," in *Proceedings of the 2006 IEEE International Conference on Web Services*, Chicago, USA, September 2006.

[2] O. Ezenwoye and S. M. Sadjadi, "Composing aggregate web services in BPEL," in *Proceedings of The 44th ACM Southeast Conference*, Melbourne, Florida, March 2006.

[3] D. Cybok, "A grid workflow infrastructure: Research articles," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1243–1254, 2006.

[4] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "Grid services for distributed system integration," *Computer*, vol. 35, no. 6, pp. 37–46, 2002.

[5] V. P. Nelson, "Fault-tolerant computing: Fundamental concepts." *IEEE Computer*, vol. 23, no. 7, pp. 19–25, 1990.

[6] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 01, no. 1, pp. 11–33, 2004.

[7] S. Goel, H. Sharda, and D. Taniar, "Message-oriented-middleware in a distributed environment." in *Third International Workshop on Innovative Internet Community Systems*, June 2003, pp. 93–103.

[8] O. Ezenwoye and S. Sadjadi, "RobustBPEL2: transparent autonomization in business processes through dynamic proxies," in *Proceedings of the 8th International Symposium on Autonomous Decentralized Systems*, Sedona, Arizona, March 2007.

[9] S. M. Sadjadi, P. K. McKinley, and B. H. Cheng, "Transparent shaping of existing software to support pervasive and autonomic computing," in *Proceedings of the first Workshop on the Design and Evolution of Autonomic Application Software 2005*, St. Louis, Missouri, May 2005.

[10] "Web services reliable messaging," http://www.ibm.com/developerworks/library/specification/ws-rm/.

[11] "WS-Reliability 1.1," November 2004, http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability%-1.1-spec-os.pdf.

[12] L. Gilman and R. Schreiber, *Distributed Computing with IBM MQSeries*. Wiley, 1996.

[13] "Microsoft. microsoft message queuing MSMQ," http://www.microsoft.com/windowsserver2003/technologies/msmq/default.ms%px.

[14] M. Hapner, R. Burridge, and R. Sharma, "Java message service specification," Sun Microsystems, Tech. Rep., Nov. 1999.

[15] S. Tai, T. Mikalsen, and I. Rouvellou, "Using message-oriented middleware for reliable web services messaging." in *Second International Workshop of Web Services, E-Business, and the Semantic Web*, 2003, pp. 89–104.

[16] V. Dialani, S. Miles, L. Moreau, D. D. Roure, and M. Luck, "Transparent fault tolerance for web services based architectures," in *Eighth International Europar Conference*. Padeborn, Germany: Springer-Verlag, aug 2002.

[17] K. P. Birman, R. van Renesse, and W. Vogels, "Adding high availability and autonomic behavior to web services." in *Proceedings of the 26th International Conference on Software Engineering*. Edinburgh, United Kingdom: IEEE Computer Society, May 2004, pp. 17–26.

[18] A. Erradi and P. Maheshwari, "wsBus: QoS-aware middleware for relaible web services interaction," in *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service*, Hong Kong, China, 2005.

[19] A. Charfi and M. Mezini, "An aspect based process container for BPEL," in *Proceedings of The First Workshop on Aspect-Oriented Middleware Developement*, Genoble, France, November 2005.

[20] L. Baresi, C. Ghezzi, and S. Guinea, "Smart monitors for composed services," in *Proceedings of the 2nd international conference on Service oriented computing*. ACM Press, 2004, pp. 193–202.

A Systematic Process for Domain Engineering

Eduardo Santana de Almeida[1], Alexandre Alvaro[1], Vinicius Cardoso Garcia[1], Daniel Lucredio[2], Renata Pontin de Mattos Fortes[2], Silvio Romero de Lemos Meira[1]
[1]*Federal University of Pernambuco, C.E.S.A.R. - Recife Center for Advanced Studies and Systems, Pernambuco, Recife, Brazil*
{esa2, aa2, vcg, srlm}@cin.ufpe.br
[2]*University of São Paulo*
{lucredio,renata}@icmc.usp.br

Abstract: Software reuse is a key aspect for improving quality and productivity. However, this process is more effective when systematically planned and managed in the context of a specific domain, where application families share some functionality. In this scenario, Domain Engineering (DE) has been seen as a facilitator to obtain the desired benefits. Nevertheless, the existing domain engineering processes present crucial problems, such as lacking of details in the three basic steps of domain engineering and not being systematic. Thus, this paper aims at defining a systematic process to perform domain engineering based on the-state-of-the-art in the area, which includes the steps of domain analysis, domain design, and domain implementation. An experimental study evaluates the viability of the use of the process and the impact of applying it to a domain engineering project.

1 Introduction

In the context of software reuse, important research including company reports [1], [2], [3], [4], informal research [5], [6] and empirical studies [7], [8], [9] have highlighted the relevance of a reuse process, once the most common way of software reuse involves developing applications reusing pre-defined assets.

However, the existing reuse processes present crucial problems [10], such as: they do not cover the three steps of domain engineering: domain analysis, design and implementation; besides, they do not define activities, sub-activities, roles, inputs, outputs of each step in a systematic way.

Thus, this paper presents a systematic software reuse process to perform domain engineering, which includes the steps of domain analysis, domain design, and domain implementation, based on a set of activities, sub-activities, inputs, outputs, principles, guidelines, and roles.

2 The Domain Engineering Process

Domain engineering is the activity of collecting, organizing, and storing past experience in building systems or parts of systems in a particular domain in the form of reusable assets (i.e. reusable work products), as well as providing an adequate means for reusing these assets (e.g. retrieval, qualification, dissemination, adaptation, assembly) when building new systems [11].

A domain engineering process should define three important steps: **Domain Analysis** (DA), **Domain Design** (DD), and **Domain Implementation** (DI). In general, the main goal of Domain Analysis is domain scoping and defining a set of reusable, configurable requirements for the systems in the domain. Next, Domain Design develops a common architecture for the system in the domain and devising a product plan. Finally, Domain Implementation implements the reusable assets, for example, reusable components, domain-specific languages, generators, and a production process [11].

The next sections presents each step in details.

2.1 The Domain Analysis Step

The term domain analysis was first introduced by Neighbors [12] as "*the activity of identifying the objects and operations of a class of similar systems in a particular problem domain*." However, neither Neighbors' nor many other works [13], [14] address the issue of "how to perform" domain analysis. These works focus on the outcome, not on the process, and success stories are more the exception than the rule.

Typically, knowledge of a domain evolves over time until enough experience has been accumulated and several systems have been implemented, so generic abstractions can be isolated and reused. In this context, our goal, in concordance with Prieto-Diaz [14], (pg. 48) is: "*to find ways to extract, organize, represent, manipulate and understand reusable information, to formalize the domain analysis process, and to develop technologies and tools to support it*."

The approach for domain analysis has three activities: **Plan Domain, Model Domain** and **Validate Domain**.

2.1.1 Plan Domain

The *first activity* in the approach corresponds to a preparation phase, to determine whether it makes good sense to invest in building a reuse infrastructure. The domain analyst collects the initial information for the subsequent steps, including: identification of the stakeholders, definition of the objectives and constraints, and market analysis. The domain analyst also collects all knowledge regarding the domain, including available documentation and existing applications.

Next, the applications to be supported by the domain are identified, including their exact scope, which features these applications support individually, and the determination of candidates for domain features. The domain analyst, together with the domain experts, develops a list of the applications of the domain, including: *existing applications* (i.e., applications that have been developed prior to the start of the domain analysis process), *future applications* (i.e., applications where the requirements are rather clear, but development has not yet started) and *potential applications* (i.e., applications for which no clear requirements exist yet, but that are seen as relevant). The list of applications also includes a list of features [15] for those applications, identified through the analysis of the applications, their documentation, and the knowledge from the domain expert.

2.1.2 Model Domain

This activity shifts attention from scoping issues to structural issues and conceptual elements within the domain. Thus, a model is developed, describing the commonality and variability within the domain. Rather than building a model for a single application in the domain, or a generic model that may be applicable at a high level to a number of applications, the domain modeling task attempts to formalize the space of variations for individual applications in the domain.

In this approach, the domain model is represented through feature models [15]. The domain analyst groups the features that were identified in the previous step in a features model, using some useful guidelines [16].

2.1.3 Validate Domain

Domain validation is achieved in five sub-activities:

A_1. *Document features*. Our approach uses the template defined by Czarnecki & Eisenecker [11]. In this template each feature consists of: Semantic description; Rationale; Stakeholders and client programs; Example applications; Constraints; Priorities; and Open/closed points in the domain;

A_2. *Check for synonyms*. Involves the analysis of each feature, in order to find and eliminate synonyms, i.e., different terms that appear to have the same domain-relevant meaning;

A_3. *Check for homonyms*. As a complementary sub-activity to the search for synonyms, it is necessary to search for homonyms, i.e., the same literal term used with different meanings in different contexts;

A_4. *Model Validation*. This sub-activity corresponds to the matching of the requirements that were expressed by stakeholders and the domain model, in order to validate its completeness and accuracy; and

A_5. *Document the domain*. In order to document the domain, the meta-model defined in [17] is used, consisting of the following information: Domain description; Domain defining rules; Exemplar system selection; Documentation; Domain Context Relationship; Domain genealogy; and Feature.

2.2 The Domain Design Step

The design step consists of seven activities, presented in the next sections. The approach is influenced by several works from the literature, such as ADD [18], UML Components [19], and the weak and strong points from reuse processes [10].

2.2.1 Module decomposition

The first step in the approach corresponds to an abstraction and decomposition phase. Initially, the domain architect chooses which domain architecture modules to decompose, usually starting with complete domain applications, which are further decomposed into subsystems and submodules.

In our approach, we still do not have a set of criteria to be used in module decomposition, as in Parnas' work [20], nor a set of rule of thumbs. However, we consider that the following issues should be balanced: *availability*, *coupling*, *extensibility*, *flexibility*, *functionality*, *information hiding*, *maintainability*, *modifiability*, *performance*, *separation of concerns*, *scalability*, *security*, and *usability*.

2.2.2 Module refinement

The module refinement is an iterative process that can be divided into three activities:

Choose the architectural drivers. According to Bass et al. [18], *architectural drivers are the combination of functional and quality requirements that "shape" the architecture or the particular module under consideration*. The drivers are found among the top-priority requirements for the module. We base the module decomposition on the architectural drivers, to reduce the problem of satisfying the most important ones. In our approach, the drivers are the requirements expressed by feature model, the quality attributes and the scenarios.

Choose the architectural patterns. Here, the domain architect selects the architectural patterns that can be applied. The patterns satisfy the architectural drivers and are constructed by composing selected tactics. Two factors guide tactic selection. The first are the drivers themselves, and the second are the side effects that a pattern

implementing a tactic has on others. Our vision of a tactic agrees with Bass et al.'s, who define it as a design decision that influences the control of a quality attribute response.

Allocate functionality using views. In this activity, the goal is to define how modules can be instantiated. The criteria are similar to those used in functionality-based design methods, such as most object-oriented design methods, but with a variation to treat features. Two approaches are proposed: **i)** *to allocate functionality based on use cases*, and **ii)** *to allocate functionality based on features.*

2.2.3 Variability representation

Variability is the ability to change or customize a system [21]. However, even with several approaches available in the literature, software architects do not have effective ways to do it [22]. In our approach, we propose the use of Design Patterns [23] to solve this problem, as already used by other works [24], [25]. Differently from these works, however, we provide guidelines for how and why each pattern should be used in each situation.

In order to design the variability of each module, we consider that it should be traceable from domain analysis assets (features) to the architecture, according to *alternative*, *or* and *optional* features [25]. Depending on the kind of association between features, different design patterns can be used. Each design pattern provides a different option for the designer, which makes the decision on which patterns to use according to some guidelines. For example, for alternative features, when a feature can be directly mapped into a single class, we suggest the *Prototype* [23], because it is simpler and allows to instantiate a specific object, depending on the feature that is used in the product, through simple inheritance. Another suggestion is to use the *Singleton* [24] pattern to keep and manage a unique instance of this class. More guidelines, for other possible situations, may be seen in [26].

2.2.4 Component grouping

This step is composed of four activities:

Measure functional dependency. The domain analyst determines the relationships between the use cases, using different metrics [26];

Cluster use cases. The domain architect defines candidate components by clustering related use cases. The clustering algorithm used for this task uses a row and column shifting method.

Allocate classes to components. Here, the domain architect locates sequence diagrams for use cases included in each component identified in the previous activity. Next, the classes participating in these sequence diagrams are assigned to the corresponding component.

Select candidate components. The domain architect, in conjunction with the project manager, identifies candidate components. The value of *t* used in the process defines the number of components and their granularity. Thus, it is recommended to apply different values of *t* to generate different clustering results and to let architects and project managers choose an optimal clustering result using the criteria. Additionally, costs and complexity can be used to select the candidate components.

2.2.5 Component identification and specification

In this step, the goal is to refine the components, including their system and business interfaces, and the core classes. For each use case, the domain analyst considers whether or not there are system responsibilities that must be modeled. If so, they are represented as one or more operations of the interfaces (just signatures). This gives an initial set of interfaces and operations.

The business interfaces are abstractions of the information that must be managed by components. Our process for identifying them is the following: *to analyze the feature model to identify classes (for each module and component); to represent the classes based on features with attributes and multiplicity; and to refine the business rules using formal language*.

After identifying the interfaces, the domain architect decides which classes from each module are in the core [19]. A core class is a business type that has independent existence within the business. The purpose of identifying core classes is to start defining which information is dependent on others, and which information can stand alone. The general rule is that we create one business interface for each core class, to manage the information represented by the core class.

2.2.6 Domain architecture representation

Once the component specification is performed, the domain architect represents the initial domain architecture based on components. Architectural views and component diagrams are used to show the components, their interconnection, and the provided and required interfaces. During this step, the domain architect can discover and refine other components, using, for example, collaboration diagrams.

2.3 The Domain Implementation Step

In our approach, we decided to use OSGi [27] to implement the components and manage their interaction and lifecycle, due to its applicability in many different scenarios, and the possibility of being used together with other technologies. This step consists of two activities: component implementation and component documentation.

2.3.1 Component implementation

In this step, the software engineer, based on requirements, implements the software components through two sets of activities, each one with a different purpose. Activities 1 to 4 deal with the *provided* services, and activities 5 to 7 deal with *required* services.

Activity 1. The first activity is to *describe the component*, providing general-purpose information, such as the vendor, version, package, among others.

Activity 2. In this second activity, the software engineer should *specify the provided services*. Artifacts developed in domain analysis and design may be reused in this activity.

Activity 3. In the third activity, the goal is to *implement the provided services*, as well as the code to register these services to be used by other components.

Activity 4. In this activity, which concludes the provided side of the component, the goal is to *build and install the component*. This involves compiling and packaging the component in a form that is suitable to be deployed.

Activity 5. In order to reuse some component, the software engineer needs to *describe the component that will reuse other services*. This is similar to Activity 1, but with the focus on the services that are required.

Activity 6. In this activity, the software engineer should *implement the connection between the required services with the rest of the code*. Here, different techniques can be employed, such as the use of adapters and wrappers, for example.

Activity 7. the last activity corresponds to *building and installing the component that reuses the services*, which is similar to Activity 4.

2.3.2 Component documentation

Most work related to component documentation [28, 29, 30, 31] are pattern-based approaches. Our step for component documentation has two important differences: i. it is based on previous works (pattern-based approaches including weak and strong points) and real world experience; ii. it follows some Principles for component documentation: *use of hypertext*; *embed content in source code*; *automation*; *leverage programming languages semantics*; *use of diagrams and figures*.

Thus, in order to document the components, a template composed of five sections is used: *Basic Information*, *Detailed Information*, *Quality Information*, *Deploy Information*, and *Support Information*. More information about the domain implementation phase can be seen in a previous work [32].

3 The Experimental Study

In order to determine whether the process meets its proposed goals, an experimental study was performed. The plan of the experiment to be presented follows the model proposed in [33], and uses the future tense, symbolizing the precedence of the plan in relation to its execution.

3.1 The Definition

Goal. To analyze the domain engineering process for the purpose of evaluating it with respect to the efficiency and difficulties of its use from the point of view of researcher in the context of domain engineering projects.

3.2 The Planning

Context. The domain engineering project will be conducted in a university laboratory with the requirements defined by the experimental staff based on real-world projects. The study will be conducted as single object study which is characterized as being a study which examines an object on a single team and a single project. All the subjects will be trained to use the process and will receive two questionnaires to provide their information (QT1) and their impression on the process (QT2).

Criteria. The benefits obtained will be evaluated quantitatively through the domain architecture and components, using the instability (*I*) [34], maintainability (*MI*) [35], and complexity (*CC*) [36] metrics. We decided to use classic Object Orientation metrics to evaluate the process because they are more well-established after years of experience with case studies and experiments. Reuse-specific metrics, although more suited to this context, still needs more experimentation and use [37]. Besides, issues such maintainability, stability and complexity have a large influence on the architecture. For example, if a component has low maintainability, it will be probably harder to reuse. So, by measuring these aspects, we are, to some extent, measuring reuse. Moreover, the difficulties will also be evaluated using qualitative data from the questionnaires.

Null Hypothesis. This is the hypothesis that the experimenter wants to reject. In this study, it determines that the use of the process does not produce benefits that justify its use and that the subjects have difficulties to apply it:

H_0: $\mu_{\text{the process generates the architecture with I >= 0.5, MI < 85, CC >= 21,}}$

The values for I, MI and CC were obtained from the literature [34, 35, 36].

Alternative Hypothesis. This is the hypothesis in favor of which the null hypothesis is rejected. In this study, the alternative hypothesis determines that the use of the process produces benefits that justify its use. Thus, the following hypothesis can be defined:

H_1: $\mu_{\text{the process generates the architecture with I < 0.5, MI >= 85, CC < 21}}$

Quantitative analysis. In this study, descriptive statistics will be used to analyze the data set [33].

Qualitative Analysis. The qualitative analysis aims to evaluate the difficulty of the application of the proposed process and the quality of the material used in the study. This analysis will be performed through questionnaire QT2.

Internal Validity. The internal validity of the study is dependent of the number of subjects. This study is supposed to have at least between seven and eight subjects to guarantee a good internal validity.

External Validity. A possible problem related to the external validity is the subjects' motivation, since some subjects can perform the study without responsibility or without a real interest in performing the project with a good quality as it could happen in an industrial project. This will be assessed through questionnaire QT2

Construct Validity. In this study, a relatively well known and easily understandable problem domain was chosen to prevent the experienced users in a certain domain to make use of it.

Conclusion Validity. This validity is concerned with the relationship between the treatment and the outcome, and determines the capability of the study to generate conclusions [33]. This conclusion will be drawn by the use of descriptive statistic.

3.3 The Project used in the Study

The project used in the experimental study was the domain engineering of the starship game domain. Three games in this domain were presented to the subjects, who had just the executables without any documentation. After performing the domain engineering, the subjects were asked to implement one application reusing the developed assets.

3.4 The Operation

Experimental Environment. The experimental study was conducted during part of a M.Sc. and Ph.D. Course in Software Reuse, during April-September 2006, at Federal University of Pernambuco. The experiment was composed of seven subjects and all the project was developed in 355 hours, 23 minutes and 57 seconds. In this project, 44 features, 33 packages, and 79 classes were created. Additionally, 5 components and 1 example application were also developed, totalizing 3638 lines of code.

Training. The subjects who used the proposed process were trained before the study began. The training took 28 hours, divided into 14 lectures with two hours each, during the course.

Subjects. The subjects were 7 MS.c. students selected by convenience sampling [33]. All the subjects had industrial experience in software development (more than one year). Three subjects had participated in industrial projects involving some kind of reuse activity, for instance, component development, framework development, or web services development. All the subjects known at least one domain analysis process (FODA); three subjects had training in conferences on some issues related to software reuse, such as design patterns and component-based development; and finally, two subjects had co-authored papers involving some aspects of software reuse.

3.5 The Analysis

Quantitative Analysis. The quantitative analysis was divided in four analyses: instability and maintainability for the architecture, complexity for the components, and the difficulties found in the analysis, design, and implementation steps. The analyses were performed using descriptive statistics. Table 1 shows the summary of the analysis.

Table 1. Results for the quantitative analysis.

Metric	Instability	Complexity	Maintainability
Mean value	0.4442	1.499	126.4058377
Max. value	0.8	1.625	150.2795238
Min. value	0.233	1.205	101.9545455
Null hypothesis	>= 0.5	>= 21	<= 85
Alternative hypothesis	< 0.5	< 21	> 85

As it can be seen, the mean values for all metrics reject the null hipothesis. Also, except for the instability metric, the maximum and minimum values still reject the null hipothesis. Thus, the results indicate that the alternative hipothesis may be true, i.e. the method helps in producing components with low complexity and high maintainability. The high instability value (0.8) was observed in only one of the components, responsible for screen management, while the other four had values below 0.5. This is expected, since in the game domain, screen management is the most intensive task that is performed, and thus this component had to be highly coupled with the others, resulting in high instability. However, if we consider the mean value, the null hipothesis is rejected, which means that the method can help to increase stability for most components.

Qualitative Analysis. After concluding the quantitative analysis, the qualitative analysis was performed. This analysis is based on the answers defined for the QT2.

Usefulness of the Process. All the subjects reported that the process was useful to perform the domain engineering project. However, four subjects indicated some improvements in domain analysis; for five subjects, some aspects in design should be reviewed; and, finally, six subjects discussed some improvements in domain implementation.

More details about the experimental evaluation may be seen in a previous work [38].

4 Related Works

In a previous work [10], eleven software reuse processes based on domain engineering (DRACO, ROSE, ODM, RSEB, FeatuRSEB, FORM) and software product lines (PuLSE, KobrA, CoPAM, PECOS, FORM's extension) are discussed, corresponding to the state-of-the-art in the area. This study shows that the processes present crucial problems such as: they do not cover the steps of domain engineering: domain analysis, design and implementation; besides, they do not define activities, sub-activities, roles, inputs, outputs of each step in a systematic way.

5 Conclusion and Future Works

Domain Engineering is a key requirement in a reuse process. However, the available reuse processes do not cover the three basic steps of domain engineering - domain analysis, domain design, and domain implementation - and neither define activities, sub-activities, roles, inputs, outputs of each step in a systematic way.

In this sense, in order to solve the problems identified in the available reuse process, this paper presented a systematic process for domain engineering, which defines a systematic way to perform it based on a set of principles, guidelines, inputs, outputs, and roles. The process is based on an extensive review of the software reuse processes, involving their weak and strong points. Additionally, an experimental study evaluated the viability of the use of the process and the impact of applying it to a domain engineering project. As future work, we are planning to improve the process with the results obtained in the experimental study and replicate it in different contexts.

More information about the domain analysis, design and implementations steps can be found in [39],[26],[40][41].

Acknowledgments

This work was developed with the financial support from CAPES and CNPq (process number: 475743/2007-5). The authors would also like to thank the people who contributed with insights to this work with discussions in worldofreuse.blogspot.com.

References

[1] D. Bauer, A Reusable Parts Center, IBM Systems Journal, Vol. 32, No. 04, September, 1993, pp. 620-624.

[2] A. Endres, Lessons Learned in an Industrial Software Lab, IEEE Software, Vol. 10, No. 05, September, 1993, pp. 58-61.

[3] M. L. Griss, Software Reuse Experience at Hewlett-Packard, 16th ICSE, Sorrento, Italy, May, 1994, pp. 270.

[4] R. Joos, Software Reuse at Motorola, IEEE Software, 1994.

[5] W. B. Frakes, S. Isoda, Success Factors of Systematic Software Reuse, IEEE Software, Vol. 12, No. 01, September, 1994, pp. 15-19.

[6] W. B. Frakes, K. C. Kang, Software Reuse Research: Status and Future, IEEE Transactions on Software Engineering, 2005.

[7] D. C. Rine, Success Factors for software reuse that are applicable across Domains and businesses, ACM Symposium on Applied Computing (SAC), 1997, pp. 182-186.

[8] M. Morisio, M. Ezran, C. Tully, Success and Failure Factors in Software Reuse, IEEE Transactions on Software Engineering, 2002.

[9] M. A. Rothenberger, K. J. Dooley, U. R. Kulkarni, N. Nada, Strategies for Software Reuse: A Principal Component Analysis of Reuse Practices, IEEE Transactions on Software Engineering, 2003.

[10] E. S. Almeida, A. Alvaro, D. Lucrédio, V. C. Garcia, S. R. L. Meira, A Survey on Software Reuse Processes, IEEE IRI, 2005.

[11] K. Czarnecki, U. W. Eisenecker, Generative Programming: Methods, Tools, and Applications, Addison-Wesley, 2000, pp. 832.

[12] J. M. Neighbors, Software Construction Using Components, Ph.D. Thesis, University of California, 1980.

[13] G. Arango, Domain analysis: from art form to engineering discipline, 5th International Workshop on Software specification and design, Pittsburgh, Pennsylvania, USA, May, 1989, pp. 152-159.

[14] R. Prieto-Diaz, Domain Analysis: An Introduction, ACM SIGSOFT Software Engineering Notes, 1990.

[15] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, SEI, Technical Report, November, 1990, pp. 161.

[16] K. Lee, K. C. Kang, J. Lee, Concepts and Guidelines of Feature Modeling for Product Line Software Engineering, 7th International Conference on Software Reuse (ICSR), 2002.

[17] K. Schmid, S. Thiel, J. Bosch, S. Johnsson, M. Jaring, B. Thomé, Scoping, Eureka ∑! 2023 Programme, ITEA project, 2001.

[18] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 2nd Edition, Addison-Wesley, 2003, pp. 560.

[19] J. Cheesman, J. Daniels, UML Component A Simple Process for Specifying Component-Based Software, Addison-Wesley, 2000.

[20] D. L. Parnas, On the Criteria to be Used in Decomposing Systems into Modules, Communications of the ACM, 1972.

[21] M. Svahnberg, J. van Gurp, J. Bosch, On the Notion of Variabilities in Software Product Lines, IEEE/IFIP WICSA, Amsterdam, Netherlands, August, 2001, pp. 45-54.

[22] J. Coplin, D. Hoffman, D. Weiss, Commonality and Variability in Software Engineering, IEEE Software, 1998.

[23] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, 1995.

[24] B. Keepence, M. Mannion, Using Patterns to Model Variability in Product Families, IEEE Software, 1999, pp. 102-108.

[25] K. Lee, K. C. Kang, Feature Dependency Analysis for Product Line Component Design, 8th ICSR, 2004, pp. 69-85.

[26] E. S. Almeida, A. Alvaro, V. C. Garcia, L. M. Nascimento, D. Lucrédio, S. R. L. Meira, Designing Domain-Specific Software Architecture: Towards a New Approach, 6th IEEE/IFIP Conference on Software Architecture (WICSA), Mumbai, India, January, 2007.

[27] OSGi Service Platform Core Specification, Core Specification, Release 4, August, 2005, pp. 276.

[28] M. F. Silva, C. M. L. Werner, Packaging Reusable Components using Patterns and Hypermedia, 4th International Conference on Software Reuse, 1996.

[29] J. Kotula, Using Patterns To Create Component Documentation, IEEE Software, 1998.

[30] A. Taulavuori, E. Niemela, P. Kallio, Component documentation—a key issue in software product lines, Journal Information and Software Technology, 2004.

[31] V. R. Basili, S. K. Abd-El-Hafiz, A Method for Documenting Code Components, Journal of Systems and Software (JSS), 1996.

[32] E. S. Almeida, E. C. R. Santos, A. Alvaro, V. C. Garcia, D. Lucrédio, R. P. M. Fortes, S. R. L. Meira. Domain Implementation in Software Product Lines Using OSGi, In the 7th International Conference on Composition-Based Software Systems (ICCBSS), Madrid, Spain, 2008.

[33] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, 2000, pp. 204.

[34] R. Martin, OO Design Quality Metrics: An Analysis of Dependencies, Consulted in June 05, 2006, Available in http://www.objectmentor.com/resources/articles/oodmetrc.pdf, October, 1994, pp.08.

[35] E. VanDoren, Maintainability Index Technique for Measuring Program Maintainability, Software Engineering Institute (SEI), Available in http://www.sei.cmu.edu/str/descriptions/mitmpm.html, Consulted in September 21, 2006.

[36] T. J. McCabe, A Complexity Measure, IEEE Transactions on Software Engineering, Vol. 02, No. 04, December, 1976, pp. 308-320.

[37] J. C. C. P. Mascena, E. S. Almeida, S. R. L. Meira, A Comparative Study on Software Reuse Metrics and Economic Models from a Traceability Perspective, IEEE IRI, 2005.

[38] E. S. Almeida, A. Alvaro, V. C. Garcia, D. Lucrédio, R. P. M. Fortes, S. R. L. Meira. An Experimental Study in Domain Engineering, In the 33rd IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering Track, Lübeck, Germany, 2007.

[39] E. S. Almeida, J. C. C. P. Mascena, A. P. C. Cavalcanti, A. Alvaro, V. C. Garcia, D. Lucrédio, S. R. L. Meira, The Domain Analysis Concept Revisited: A Practical Approach, 9th ICSR, 2006.

[40] E. S. Almeida, E. C. R. Santos, A. Alvaro, V. C. Garcia, D. Lucrédio, R. P. M. Fortes, S. R. L. Meira, A Method for Domain Implementation in Software Product Lines, TOOLS Conference, Switzerland, 2007, in evaluation.

[41] E. S. Almeida, A. Alvaro, V. C. Garcia, L. M. Nascimento, D. Lucrédio, S. R. L. Meira. A Systematic Approach to Design Domain-Specific Software Architectures , Journal of Software, Academy Publisher, Vol. 02, No. 02, August, 2007, pp 38-51.

Diagnosing Runtime Violations of Security & Dependability Properties

Theocharis Tsigritis and George Spanoudakis
Department of Computing, City University London, UK
{t7t, G.Spanoudakis}@soi.city.ac.uk

Abstract

Monitoring the preservation of security and dependability (S&D) properties of complex software systems is widely accepted as a necessity. Basic monitoring can detect violations but does not always provide sufficient information for deciding what the appropriate response to a violation is. Such decisions often require additional diagnostic information that explains why a violation has occurred and can, therefore, indicate what would be an appropriate response action to it. In this paper, we describe a diagnostic procedure for generating explanations of violations of S&D properties developed as part of a runtime monitoring toolkit, called EVEREST. The procedure is based on a combination of abductive and evidential reasoning about violations of S&D properties which are expressed in Event Calculus.

1. Introduction

Monitoring security and dependability (S&D) properties of software systems at runtime is widely accepted as a measure of increased resilience to dependability failures and security attacks, and several approaches have been developed to support it (see [5] for a survey). Whilst basic monitoring provides the core functionality for detecting violations of such properties, it cannot always provide the information that is necessary in order to understand the reasons that underpin the violation of a property and decide what would be an appropriate reaction to it.

To appreciate the problem, consider the case of an *Air Traffic Management System* (ATMS), which uses radars to monitor trajectories of airplanes in different air spaces. The operations of ATMS may be monitored at runtime to ensure the availability and integrity of its components (e.g. radars), and the information generated by and exchanged between them. An example of a property that can be monitored in ATMS is a property requiring that in cases where there are more than one radars covering a particular airspace and one of these radars sends a signal indicating that an airplane is in the relevant airspace, every other radar that covers the same space should also send a signal indicating the presence of the plane in the particular airspace within a certain time period after the receipt of the initial signal.

In cases where this property is violated, detecting the occurrence of the violation is not sufficient for establishing the reasons why some radar has sent a signal but another other has not. Getting diagnostic information about these reasons is necessary for taking appropriate action as the violation may have been due to different reasons, some of which are listed below:

(i) The radar that did not send the expected signal was malfunctioning.
(ii) The communication link between the radar that did not send the expected signal and the monitor was malfunctioning or an intruder captured the signal and prevented it from reaching the monitor.
(iii) The radar that sent the expected signal was malfunctioning or its identity was faked by an intruder who sent a fake signal to the monitor.

Thus, identifying the reason for the violation is important for taking one or more actions that could restore the integrity of the operation of ATMS.

In this paper, we present a diagnosis system that we have developed as part of a monitoring framework [13], called EVEREST (*EVE*nt *REaS*oning *T*oolkit). This framework has been developed within the European integrated research project SERENITY to support the monitoring of S&D properties in distributed and dynamically evolving systems. EVEREST supports the specification and monitoring of properties expressed in Event Calculus (EC) [12] as rules.

In this paper, we present an extension of this framework supporting the generation of diagnostic information for violations of S&D properties. The provision of diagnostic information is based on the generation of alternative *explanations* for the events which are involved in the violations of rules, and the assessment of the plausibility of these explanations based on whether their effects correspond to events recorded during the operation of the monitored system. The key characteristic of our approach is the use of abductive reasoning [1][9] for the generation of explanations and belief based reasoning [11] for the assessment of explanation plausibility.

The rest of this paper is structured as follows. In Section 2, we provide a brief overview of EVEREST. In Section 3, we describe the different stages of the diagnostic process. In Section 4, we overview related work and, finally, in Section 5, we present conclusions and directions for future work.

2. Monitoring framework

EVEREST is a monitoring toolkit which consists of a generic engine for checking violations of properties expressed as EC rules of the form $body \Rightarrow head$. The

semantics of a rule is that when its body evaluates to *True*, its head must also evaluate to *True*. EC is a first-order metric temporal logic language which can be used for representing and reasoning about *events* and their effects on the state of a system over time. EVEREST rules are defined in terms of the standard EC predicates. These include the predicates (i) *Happens(e,t,ℜ(lb,ub))* which denotes that an instantaneous event *e* occurs at some time *t* within the time range *ℜ(lb,ub)*, (ii) *HoldsAt(f,t)* which denotes that a state (a.k.a fluent) *f* holds at time *t,* (iii) *Initiates(e,f,t)* and *Terminates(e,f,t)* which denote the initiation and termination of a fluent *f* by an event *e* at time *t* respectively, and (iv) *Initially(f)* which denotes that a fluent holds at the start of the operation of a system.

An example of an EVEREST rule is:

Rule 1: **Happens**(signal(_r1, _a, _s),t1,R(t1,t1) ∧ **HoldsAt**(covers(_r1,_s),t1) ∧ (∃_r2) **HoldsAt**(covers(_r2,_s), t1) ⇒ **Happens**(signal(_r2,_a,_s), t2 ,R(t1, t1+5))

This rule expresses the condition about the radars of ATMS that we discussed in the introduction. More specifically, Rule-1 will be violated if the monitor receives a *signal* event by one of the radars of ATMS that cover a specific airspace but not the other.

3. Diagnostic process

As shown in Figure 1, the overall process of diagnosing the causes of rule violations includes four stages. These stages are:

(1) The *explanation generation* stage in which all the *possible explanations* for the individual events that have caused the violation (referred to as "violation observations" henceforth) are generated.

(2) The *explanation effect identification* stage in which the possible consequences (effects) of the explanations of violation observations are derived.

(3) The *plausibility assessment* stage in which the effects of explanations are checked against the event log of the monitor to see if there are events that match and, therefore, provide supportive evidence for them.

(4) The *diagnosis generation* stage in which an overall diagnosis for a violation is generated based on the derived explanations

The generation of explanations and their effects in stages (1) and (2) above is based on a model of the behaviour of the monitored system that is expressed in Event Calculus by formulas called *assumptions*. In the following, we discuss each of the above stages in detail.

3.1 Explanation generation

The generation of explanations for violation observations is based on abductive reasoning. More specifically, given a set *Ω* of events and fluents that are involved in the violation of a monitoring rule, this stage of the diagnostic process tries to find a set of *explanation formulas Φ* which, in conjunction with the set of the *assumptions* about the system that is being monitored and the events reported to the monitor by the time at which the explanation is required (collectively referred to as *TH theory* in the following), entail *Ω*. Formally, this is a search for a set of atomic formulas Φ that satisfy the conditions:

(Cnd 1): TH ∪ Φ |- Ω, and

(Cnd 2): ∀f in Φ: predicate (f) ∈ A-Preds

where *predicate(f)* is the predicate of formula *f* and *A-Preds* is a set of abducible predicates whose truth value can be established only by abductive reasoning.

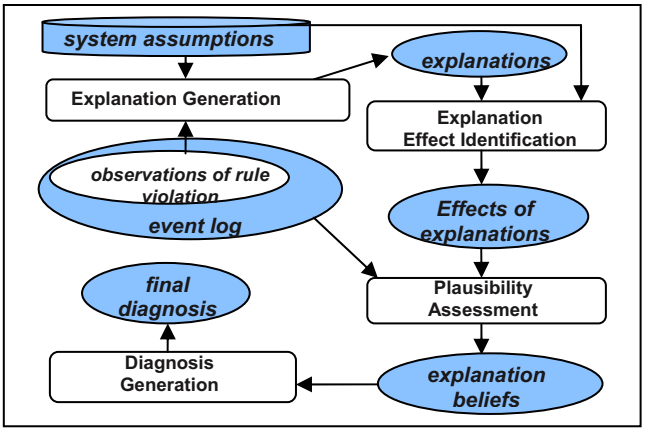


Figure 1. Diagnostic process

The search for explanations is based on a newly developed algorithm (see [13]) which starts from a violation observation *P* that needs to be explained and tries to find all assumptions of the form *a:* $B_1 \wedge ... \wedge B_n \Rightarrow H$ in *TH* whose head *H* can be unified with *P*. When such an assumption is found, the algorithm checks if: (i) the unification of *P* with *H* provides concrete values for all the non time variables of the predicates $B_1, ..., B_n$ in its body, and (ii) it is possible to derive concrete time ranges for the time ranges of all these predicates. If these conditions are satisfied, the algorithm instantiates the predicates B_1, …, B_n and identifies which of these predicates are observable predicates (*O-preds*), deducible predicates (*D-preds*) or abducible predicates (*A-preds*), assuming that these are disjoint categories of predicates.

Then, the algorithm checks if each of the *O-Preds* and *D-preds* in the body of *a* can be matched with some recorded event or derived from the events in the monitor's log and the known system assumptions, respectively. If there are *O-preds* and *D-preds* that cannot be verified via this check, the algorithm tries to find an abduced explanation for them recursively. If such explanations are found for all the non verified *O-preds* and *D-preds*, these explanations along with the *A-preds* that were determined in the current step of the explanation process are reported as the possible

explanation of the initial violation observation *P*. In cases, however, where there are *O-Preds* or *D-preds* in the body of *a* that can neither be verified nor explained by abduction, the explanation generation path using *a* will fail.

As an example of explanation generation, consider again *Rule 1*. This rule would be violated by the event (E7) in the log of Figure 2 (*Happens(signal(R1,A1,S1),7,R(7,7))*) and the predicates ¬*Happens(signal(R2,A1,S1),t,R(7,12))*, *HoldsAt(covers(R1,S1),7)* and *HoldsAt(covers(R2,S1),7)* which can be derived from this log. More specifically, the predicate ¬*Happens(signal(R2,A1,S1),t,R(7,12))*, which denotes the absence of a signal from radar *R2* from T=7 to T=12 is deduced by the principle of *negation as failure* (NF) from the events (E4) and (E8) that the monitor receives from the radar *R2* at T=1 and T=13. This deduction is possible as soon as the monitor receives (E8) since this event indicates that the time of R2 when it was sent was T=13 and the monitor had received no other event from *R2* since receiving the event (E4) at T=1. Also the predicates *HoldsAt(covers(R1,S1), 7)* and *HoldsAt(covers(R2,S1), 7)* can be deduced from events (E1) and (E2) in Figure 2, which denote that the radars R1 and R2 cover the airspace S1 initially, and the absence of any event signifying the repositioning of any of these two radars until T=7 when the monitor receives the signal for the presence of the aircraft A1 in S1 from R1 (this deduction is based on the axioms of EC [12]).

(E1) **Initially**(covers(R1,S1),0) [captor-0]
(E2) **Initially**(covers(R2,S1),0) [captor-0]
(E3) **Happens**(changeOfLandingApproach(AR-a,S2),0,R(0,0)) [captor-AR-a]
(E4) **Happens**(signal(R2,A2,S2),1, R(1,1)), [captor-R2]
(E5) **Happens**(changeOfLandingApproach(AR-a,S1),2,R(2,2)) [captor-AR-a]
(E6) **Happens**(permissionRequest(A1,S1),3,R(3,3)) [captor-0]
(E7) **Happens**(signal(R1,A1,S1),7,R(7,7)) [captor-R1]
(E8) **Happens**(signal(R2,A5,S1),13,R(13,13)) [captor-R2]

Figure 2. Event log

To explain the violation of Rule-1 in our example, the predicates *Happens(signal(R1,A1,S1),7,R(7,7))* and ¬*Happens(signal(R2,A1,S1),t,R(7,12))* which are involved in the violation need to be explained individually. Assuming the following assumptions about ATMS,

(A0) **Initiates**(_e1,_f),t1,R(t1,t1)) ∧ ¬∃_e2,t2: **Terminates**(_e2,_f),t2,R(t1,t2)) ⇒ **HoldsAt**(_f,t2)
(A1) **Happens**(inspace(_a,_s),t1,R(t1,t1)) ∧ **HoldsAt**(covers(_r,_s),t1) ⇒ **Happens**(signal(_r,_a,_s),t2, R(t1,t1+5))
(A2) **Happens**(inspace(_a,_s),t1, R(t1,t1)) ⇒ **Happens**(permissionRequest(_a,_s), t2, R(t1-20,t1-1))

the search for an explanation of *Happens(signal(R1,A1,S1),7,R(7,7))* detects that this predicate can be unified with the predicate *Happens(signal(_r,_a,_s), t2, R(t1,t1+5))* in the head of assumption (A1). The unification of these two predicates will be *{_r/R1, _a/A1, _s/S1}* and the linear constraint system generated for the time variable t1 in (A1) will include the constraints $t_1 \leq 7$ and $7 \leq t_1 + 5$. Thus, as the non time variables in the body of (A1) are covered by the unification and the constraints $t_1 \leq 7$ and $7 \leq t_1 + 5$ determine a feasible time range for t1 (i.e., [2,7]), the conditions of the explanation generation process are satisfied and the predicate *Happens(inspace(A1,S1),t1,R(2,7))* will be generated as a possible explanation of *Happens(signal(R1,A1,S1),7,R(7,7))*. Also, assuming that *Happens(inspace(_a,_s),t1,R(t1,t1))* belongs to the set of the abducible predicates *A-preds*, there will be no need for seeking for more refined explanations of it.

Note, however, that as *Happens(inspace(A1,S1),t1,R(2,7))* has been generated from assumption (A1), it can be returned as an explanation only if the other instantiated predicate of the body of (A1), namely *HoldsAt(covers(R1,S1),t1)*, is *True* when t1 takes values in the range R(2,7). The latter predicate, however, can be deduced from the log of Figure 2 and assumption (A0). Thus, *Happens(inspace(A1,S1),t1,R(2,7))* becomes a possible explanation of *Happens(signal(R1,A1,S1),7,R(7,7))*.

3.2 Explanation effect identification

Following the generation of explanations, the second stage of the diagnosis process is the identification of their expected effects. These effects are needed in order to assess the plausibility of explanations.

The assessment of explanation plausibility is based on the hypothesis that if the expected effects of an explanation match with events which have occurred and recorded during the operation of the system that is being monitored, then there is supportive evidence for the explanation. This is because the events that match the expected effects of an explanation might also have been caused by it.

The identification of the expected effects of explanations is based on deductive reasoning. More specifically, given an explanation $Exp=P_1 \wedge ... \wedge P_n$ that is expressed as a conjunction of abduced atomic predicates, the diagnosis process iterates over its constituent predicates P_i and, for each of them, it finds the system assumptions $B_1 \wedge ... \wedge B_n \Rightarrow H$ that have a predicate B_j in their body which can be unified with P_i and the rest of the predicates B_u (u=1,…,n and u≠j) *True*. For such assumptions, if the predicate *H* in the head of the assumption is fully instantiated and its time range is determined, *H* is derived as a possible consequence of P_i. Then, if *H* is an observable predicate, i.e., a predicate that can be matched with recorded events, *H* is added to the

expected effects of *Exp*. If *H*, however, is not an observable predicate, the effect identification process tries to generate the consequences of *H* recursively and, if it finds any such consequences that correspond to observable events, it adds them to the set of the expected effects of *Exp*. In this way, the diagnosis process computes the transitive closure of the effects of *Exp*.

To clarify this stage of the diagnosis process, consider again the ATMS system and suppose that, in addition to assumptions (A1) and (A2), three more assumptions are known for ATMS, namely:

(A3) **Happens**(inspace(_a,_s),t1,R(t1,t1)) ⇒
Initiates(inspace(_a,_s), inairspace(_a,_s),t1)

(A4) **Initiates**(inspace(_a,_s), inairspace(_a,_s),t1) ∧
HoldsAt(landing_airspace_for(_s,_arpX),t1) ⇒
Happens(landingRequest(_a, _arpX), t2, R(t1-10,t1))

(A5) **Happens**(changeOfLandingApproach(_arpX,_s),t1,R(t1,t1))
⇒ **Initiates**(changeOfLandingApproach(_arpX,_s),
landing_airspace_for(_s,_arpX),t1)

The first of these assumptions (A3) states that when an event that signifies the entrance of an aircraft *_a* in an airspace *_s* becomes known a fluent called *inairspace(_a,_s)* should be initiated to signify the presence of *_a* in *_s* unless this fluent already holds. The second assumption (A4) states that when an aircraft *_a* enters an airspace *_s* which is used as the landing route for approaching an airport *_arpX*, then the aircraft *_a* must have made a landing request for the particular airport within the last 10 time units before entering *_s*.

Using (A3) and (A4), it is possible to determine the expected effects of the predicate *Happens(inspace(A1,S1),t1,R(2,7)))* that was generated as a possible explanation of *Happens(signal(R1,A1,S1),7,R(7,7))*. Specifically, assuming that the airspace *S1* is the landing airspace of an airport *AR-a* then the entrance of the aircraft *A1* into *S1* should have been preceded by some request from *A1* to land in *AR-a* or, equivalently, that an event *Happens(landingRequest(A1,AR-a), t2, R(0,6))* should have occurred. Thus, the latter runtime event is an expected effect of the explanation *Happens(inspace(A1,S1),t1,R(2,7))*.

Formally, from *Happens(inspace(A1,S1),t1,R(2,7)))* and the assumption (A3) the predicate *Initiates(inspace(A1,S1), inairspace(A1,S1), t1)* can be derived for *t1* in [2,7]. As the latter predicate, however, is not an observable predicate, the diagnosis process will try to identify whether it has any observable consequences of its own. Whilst searching for such consequences, *Initiates(inspace(A1,S1), inairspace(A1,S1), t1)* can be unified with the first predicate in the body of (A4). Furthermore, the other predicate in the body of this assumption, namely the predicate *HoldsAt(landing_airspace_for(S2,AR-a), t)* can also be deduced to be *True* for the time range [2,7] (i.e., for t in [2,7]) from the event (E5) in Figure 2 and the assumptions (A5) and (A0). Thus, both predicates in the body of (A4) are *True* and, therefore, the predicate *Happens(landingRequest(A1,AR-a), t2, R(0,6))* in its head can be derived from it. Assuming that *landingRequest(_a, _arpX)* is an observable event, *Happens(landingRequest(A1,AR-a), t2, R(0,6))* will be established as an expected effect of the explanation *Happens(inspace(A1,S1),t1,R(2,7)))*.

3.3 Assessment of explanation plausibility

After deriving the expected effects $\Phi^C=\{C_1,...,C_L\}$ of an explanation Φ, the diagnosis process searches the event log to find events that can match these effects. In this search, a match between an event *e* in the log, which has been produced by an event captor *Captor(e)* and has a timestamp t_e, and an effect C_k (k=1,…,L) is detected only if: (i) *e* has been produced by the same event captor as the captor that C_k is expected to be produced from, (ii) *e* can be unified with C_k , and (iii) the timestamp of *e* falls within the time range of C_k.

Whilst the presence of a matching event for an expected effect of an explanation confirms that the effect has indeed occurred and casts some positive evidence in the validity of the explanation, the absence of a matching event for an effect at the time of the search does not necessarily mean that such an event has not occurred. This is because, although an event that satisfies the conditions (i)−(iii) above may have occurred, this event might not have arrived yet at the event log of the monitoring framework due to communication delays in the "channel" between the event captor that captured it and the framework. To cope with this problem, the search for events that match an explanation effect C_k establishes that no such events have occurred if at the time of the search there is no event *e* satisfying the conditions (i)-(iii) above, and the last known value of the clock of $Captor(C_k)$ (i.e., the timestamp of the last event in the log that has arrived at the monitor from this captor) is greater than the upper boundary of the time variable of C_k.

Note, however, that even with the above search condition, there is a possibility of having effects C_k for which, although no matching event satisfying (i)-(iii) can be found at the time of the search, the last received event from the relevant captor has a timestamp that is less than or equal to the upper time boundary of C_k. Such effects cannot be confirmed or disconfirmed and, therefore, cast positive or negative evidence for Φ. To cope with this uncertainty, we use the *Dempster Shafer (DS) theory of evidence* [11] for the assessment of the plausibility of an explanation, and define the function that gives the basic probability assignment to the validity of explanations as follows:

Definition 1: The basic probability of an explanation validity is computed by the function:

$m_E(Valid(\Phi)) = |\Phi^{C+}| / |\Phi^C|$

$m_E(\neg Valid(\Phi)) = |\Phi^{C-}| / |\Phi^C|$

$m_E(Valid(\Phi) \vee \neg Valid(\Phi)) = |\Phi^C - (\Phi^{C+} \cup \Phi^{C-})| / |\Phi^C|$

where

- Φ^{C+} is the set of confirmed effects of Φ, defined as $\Phi^{C+} = \{C_k \mid C_k \in \Phi^C$ **and** $\exists e.\ (e \in Log$ **and** $Captor(e) = Captor(C_k)$ **and** $t_k^{LB} \leq t_e$ **and** $t_e \leq t_k^{UB}$ **and** $unifier(e, C_k) \neq \emptyset)\}$
- Φ^{C-} is the set of a set of disconfirmed effects of Φ, defined as $\Phi^{C-} = \{C_k \mid C_k \in \Phi^C$ **and** $\neg\exists e.\ (e \in Log$ **and** $Captor(e) = Captor(C_k)$ **and** $t_k^{LB} \leq t_e$ **and** $t_e \leq t_k^{UB}$ **and** $unifier(e, C_k) \neq \emptyset)$ **and** $lastTime(Captor(C_k)) > t_k^{UB}\}$
- t_k^{LB}, t_k^{UB} are the lower and upper boundaries of the time range of C_k, t_e is the timestamp of the event *e,* and *lastTime(Captor(C_k))* is the timestamp of the last event arrived from *Captor(C_k)* to the monitor.

According to this definition, the probability of the validity of an explanation Φ is measured as the ratio of its effects that have been confirmed by events in the monitor's log to all its effects. Also the probability of an explanation Φ not being valid is measured as the ratio of the effects of Φ that have been disconfirmed by events in the log to all its effects. Note that, in general it will be $\Phi^{C+} \cup \Phi^{C-} \subseteq \Phi^C$, and therefore $m_E(Valid(\Phi)) + m_E(\neg Valid(\Phi)) \leq 1$. Thus, m_E is not a classic probability function. As we prove in [13], however, m_E satisfies the axioms of *basic probability assignments* in the DS theory of evidence.

As an example of applying m_E, consider the estimation of the basic probability of the explanation *Happens(inspace(A1,S1),t1,R(2,7))* of the violation observation *Happens(signal(R1,A1,S1),7,R(7,7))* of *Rule-1*. As we discussed in Section 3.2, an expected effect of this explanation is *Happens(landingRequest(A1,AR-a),t2,R(0,6))*. Another expected effect of it is the predicate *Happens(permissionRequest(A1,S1),t2,R(0,7))*. The latter effect can be derived from assumption (A2). According to this assumption, an aircraft that enters a particular airspace at some time point, should have requested permission to do so within 20 time units prior to its entrance.

Thus, assuming that the request for diagnosing the violation of *Rule-1* is made at *T=15*, a search in the event log of Figure 2 will identify that the event *Happens(permissionRequest(A1,S1),3,R(3,3))* provides confirmatory evidence for *Happens(permissionRequest(A1,S1),t2,R(0,7))* but there is no matching event for *Happens(landingRequest(A1,AR-a),t2,R(0,6))*.

Furthermore, if *Happens(landingRequest(A1,AR-a), t2, R(0,6))* refers to events which are captured and transmitted by the event captor *captor-AR-a* then at the time of the search (*T=15*), it will not be possible to establish whether an event matching *Happens(landingRequest(A1,AR-a),t2,R(0,6))* has occurred. This will be so because, as shown in Figure 2, the last event received from *captor-AR-a* until *T=15* is *Happens(changeOfLandingApproach(AR-a,S1),2,R(2,2))* and, therefore, the latest known time for this captor (*lastTime(captor-AR-a)*)) is 2. Thus, the basic probability in the validity of the explanation Φ=*Happens(inspace(A1,S1),t1,R(2,7))* will be: $m_E(Valid(\Phi)) = 1/2 = 0.5$, $m_E(\neg Valid(\Phi)) = 0/2 = 0$ and $m_E(Valid(\Phi) \vee \neg Valid(\Phi)) = 1/2 = 0.5$.

3.4 Diagnosis generation

Having obtained the basic probabilities in the validity or not of individual explanations, the fourth stage in the diagnosis process is to construct an aggregate explanation of the S&D rule violation. The construction of such aggregate explanations is based on assessing the overall belief in the *genuineness* of the events which are involved in the violation. This assessment is based on the hypothesis that an event *E*, which is involved in a violation of an S&D rule, is genuine if and only if at least one of the explanations that have been generated for it is valid.

Based on this hypothesis, as we show in [13], the belief in the genuineness or not of *E* (*Gen(E)* and $\neg Gen(E)$, respectively) can be measured by the functions:

- $Bel(Gen(E)) = Bel(\vee_{i=1,\ldots,n} Valid(\Phi_i)) = \Sigma_{I \subseteq \{1,\ldots,n\} and\ I \neq \emptyset} (-1)^{|I|+1} \{\Pi_{i \in I} m_E(Valid(\Phi_i))\}$ (F2)
- $Bel(\neg Gen(E)) = Bel(\wedge_{i=1,\ldots,n} \neg Valid((\Phi_i)) = \Pi_{i=1,\ldots,n} m_E(\neg Valid(\Phi_i))$ (F3)

where Φ_i (i=1,…,n) are the alternative explanations of *E*.

The beliefs computed by the above formulas are used to decide if a violation observation *E* is confirmed by its available explanations. More specifically, a violation *E* is confirmed only if $Bel(Gen(E)) > Bel(\neg Gen(E))$. In cases, however, where no explanation can be generated for a violation observation, the diagnosis process attempts to find an explanation of its negation and, if this is possible, the beliefs in the genuineness of the event are calculated by the formulas:

- $Bel(Gen(E)) = 1 - Bel(Gen(\neg E))$ (F4)
- $Bel(\neg Gen(E)) = Bel(Gen(\neg E)))$ (F5)

Table 1. Beliefs in violation observations of Rule -1

Event (e)	Bel(Gen(e))	Bel(¬ Gen(e))	Confir med
P1=Happens(signal(R1,A1,S1, 7, R(7,7)))	0.5	0	YES
P2=HoldsAt(covers(R1,S1),7)	–	–	YES
P3= HoldsAt(covers(R2,S1),7)	–	–	YES
P4=¬Happens(signal(R2,A1,S 1), t, R(7,12))	0.5	0.5	NO

Using (F2)-(F5), the beliefs in the genuineness of the predicates involved in the violation of Rule-1 above are as shown in Table 1. The beliefs in this table are calculated from the alternative explanations of the

relevant violation observations. More specifically, for the predicate *P1=Happens(signal(R1,A1,S1),7,R(7,7)))* there is a single explanation P_{11}=*Happens(inspace(A1,S1),t1,R(2,7))* with basic probabilities $m_E(Valid(P_{11}))\}=0.5$ and $m_E(\neg Valid(P_{11}))\}=0$, as we discussed earlier. Thus, $Bel(Gen(P1))=m_E(Valid(P_{11}))\}=0.5$ and $Bel(\neg Gen(P1))=m_E(\neg Valid(P_{11}))\}=0$. The predicates *P2=HoldsAt(covers(R1,S1),7)* and *P3=HoldsAt(covers(R2,S1),7)* are also confirmed without using beliefs measures, as they are both derived from the runtime events (E1) and (E2) in Figure 2. Finally, *P4* is a negated predicate and, since no explanation of it can be generated from the assumptions of ATMS, the diagnosis process generates explanations of its non-negated form *Happens(signal(R2,A1,S1),t,R(7,12))*. Following the same reasoning process as in the case of *P1*, *P41=Happens(inspace(A1,S1,t,R(7,17))* will be derived as an explanation of $\neg P4$ with basic probabilities $m_E(Valid(P41))\} = 0.5$ and $m_E(\neg Valid(P41))\} = 0$. Thus, $Bel(Gen(\neg P4))=0.5$ and $Bel(\neg Gen(\neg P4))= 0$ and, from (F4) and (F5), $Bel(\neg Gen(P4))=0.5$ and $Bel(Gen(P4))= 0.5$. Thus, *P4* is reported as an unconfirmed predicate.

4. Related work

In the context of software system monitoring, diagnosis focuses on the detection of the reasons for system failures and typically involves the identification of trajectories of system events that have led to a failure (problematic event) using automata that recognize faulty behaviour [4][6][10]. In [4], diagnosis is carried through the synchronization of automata modelling the expected behaviour of a monitored system and the events captured from it. In [6] a similar but incremental approach is taken where synchronization is performed for individual system components and then aggregated for the global system.

Our approach is different from the above, as our focus is not the detection of the cause of faulty behaviours (this is the subject of earlier work described in [13]) but the explanation of such causes in the presence of incomplete and/or not trusted event traces. Our approach draws upon work on temporal abductive reasoning [1][3][8][12] and its applications to diagnosis [2][7], but is based on a newly developed algorithm for abductive search with EC that generates all the possible explanations of a formula (unlike [3][12]) and computes beliefs in explanations using the DS theory.

5. Conclusions

In this paper, we have presented the extension of a framework supporting the runtime monitoring of software systems. The extension has been introduced to provide diagnostic information when violations of monitored properties occur. The provision of diagnostic information is based on alternative *explanations* of events involved in violations of properties generated by abductive reasoning using a model of the behaviour of the monitored system expressed in Event Calculus. Our approach supports also the computation of beliefs in the plausibility of explanations based on evidence about their expected effects that is gathered from the event log of the monitored system. A more detailed account of our approach and its implementation is given in [14].

Currently, we are conducting an experimental evaluation of it in the context of industrial case studies of the SERENITY project.

6. Acknowledgements

This work has been partially funded by the European integrated research project SERENITY (FP6-IST-2006-27587).

7. References

[1] Console L. et al. “Local Reasoning and Knowledge Compilation for Efficient Temporal Abduction”. IEEE Trans. on Knowledge & Data Engineering 14(6): 1230-1248, 2002

[2] De Kleer J., Williams B.C. “Diagnosing Multiple Faults”, Artif. Intell. 32(1): 97-130, 1987

[3] Denecker M. et al. “Temporal reasoning with abductive event calculus”, 10th ECAI, 1992.

[4] Grastien A., Cordier M., Largouët C., “Incremental Diagnosis of Discrete-Event Systems”, 15th Int. Work. On Principles of Diagnosis (DX05), 2005

[5] Lazarevic A., Kumar V., Srivastava J. “Intrusion detection: a survey”, In Managing cyber-threats: issues approaches & challenges, Springer. 2005

[6] Pencolé Y., Cordier M. “A formal framework for the decentralised diagnosis of large scale discrete event systems & its application to telecommunication networks”, Artif. Intell. 164: 121-180, 2005

[7] Poole D. “Explanation and prediction: an architecture for default and abductive reasoning”, Comp. Intell. 5(2): 97-110, 1989

[8] Ray O., Kakas A. “ProLogICA: a practical system for Abductive Logic Programming”. 11th Int. Works. on Non-monotonic Reasoning, 304-312, 2006

[9] Reiter R. “A theory of diagnosis from first principles”, Artif. Intell. 32(1): 57-96, 1987.

[10] Sampath M. et al. “Failure diagnosis using discrete-event models”, IEEE Trans. on Control Systems Technology, 4(2):105-124, 1996.

[11] Shafer G. “A Mathematical Theory of Evidence”, Princeton University Press, 1975.

[12] Shanahan M. “Abductive Event Calculus Planner”, J. Logic Programming 44: 207-239, 2000.

[13] Spanoudakis G., Mahbub K., “Non intrusive monitoring of service based systems”. Int. J. of Cooperative Inform. Systems, 15(3):325–358, 2006.

[14] Spanoudakis G., Tsigritis T. “v1 of diagnosis prototype”. Deliverable A4.D5.1, SERENITY Project, http://www.serenity-forum.org/ 2008.

Translating Workflow Diagrams into Web Designs

Antonio Navarro[1], Jorge Merino[2], Alfredo Fernández-Valmayor[1], Jesús Cristobal[2]
[1]*DISIA, C/ Profesor José García Santesmases s/n, 28040, Madrid*
[2]*UATD, Edificio Jardín Botánico, Avda. Complutense s/n, 28040, Madrid*
anavarro@sip.ucm.es, jmerino@pas.ucm.es, alfredo@sip.ucm.es, jcristobal@pas.ucm.es

Abstract

This paper introduces a design method that permits the translation of workflow diagrams into detailed web design diagrams. This detailed design, defined in terms of UML WAE class and sequence diagrams, describes the architectural design of the presentation tier and its interaction with the business tier of the web application. The main benefit of this approach is an improvement in the comprehensibility and maintainability of web applications, because part of the detailed architectural design is obtained from workflow diagrams. In addition, a model-driven architecture approach is enabled.

1. Introduction

Design is one of the biggest concerns in web engineering [18]. Thus, at present, there is a wide array of design notations that can be used in order to characterize the design of web applications [6, 7, 10, 11, 12, 16, 25]. Although most of these notations were originally intended to characterize websites with marginal presence of business logic [4], at present, these design notations are also able to specify the business processes of web applications [2, 4, 6, 7, 15, 16, 24]. These notations use different types of modeling diagrams to characterize different aspects of web applications: (i) *structural diagrams* to characterize the data tier; (ii) *navigational diagrams* to characterize the navigational maps; (iii) *user interface diagrams* to characterize the layout of the user interface; and (iv) *dynamic diagrams*, to characterize the business processes. Regarding dynamic diagrams, several design notations use *workflow diagrams* [2, 4, 15], or other types of flow-based notations [3, 24, 28] as high level artefacts that describe the business processes of web applications. In most cases, these workflow diagrams (and the rest of diagrams of the design notation) are high-level modeling artefacts that, using CASE tools specifically designed for concrete design notations, can be semi-automatically translated into running applications [4, 9, 14, 15]. In other cases, where no CASE tools are provided, the transition from high-level design to architectural design is not described.

Therefore, in our opinion, the presence of workflows as high-level modeling artefacts can introduce some complications in the life cycle of a web application, because: (i) if a proprietary CASE is used to semi-automatically produce the web application, the detailed architectural design of the application may be left out. Thus, the maintenance and adaptation of this application to users' demands may be compromised, because the whole developing effort is tied to a specific CASE tool and its design notation and features; (ii) if no CASE tool is provided, the transition from a high-level design to a detailed architectural design is far from straightforward and may be poorly documented. This, in turn, could compromise the life cycle of the application: its development, maintenance and adaptation to user demands.

In this paper we propose to overcome these difficulties by introducing a simple interpretation for workflow diagrams based on the *UML Web Application Extension* (UML WAE) class and sequence diagrams [7]. By including some stereotypes in the workflow actions, and selecting a concrete web architecture (i.e. Model 1, Model 2 or multitier [1, 5]), the architectural design of the presentation tier and its interaction with the business tier of the web application [1] can be obtained. The approach presented in this paper is the result of the modeling effort made during the development, maintenance and adaptation to users' demands at the *Universidad Complutense de Madrid Virtual Campus* (UCM Virtual Campus) [26]. This web application provides students, teachers and researchers with all the support that modern information and communications technologies can provide to improve the quality of learning and research activity at the university [17]. Thus, the application

allows the creation of on-line e-learning courses, their combination, the creation of virtual spaces for research seminars, and many other virtual workspaces demanded by its users.

2. Translation from Workflow Diagrams to UML Diagrams

At present, *UML activity diagrams* [23] and *Business Processes Modeling Notation* (BPMN) [19] are some of the most successful workflow-based notations used to specify the business processes of web applications [2, 4, 15]. Although both notations are very similar [27], in this paper we use UML activity diagrams because they are the modeling diagrams integrated in most UML CASE tools and because they are used in several web engineering design notations [2, 15]. UML 2 activity diagrams can include a great number of components [23]. Therefore, a translation of every component of UML 2 activity diagrams into UML WAE diagrams is outside the scope of this paper. Instead, a translation for the main elements of this type of diagram, in terms of Model 2 architecture, is provided.

UML WAE notation considers the principle of separation of concerns [7]. According to this principle: (i) web pages executed in the server are UML classes stereotyped with the `server page` stereotype; (ii) web pages presented to the client are UML classes stereotyped with the `client page` stereotype; (iii) forms are UML classes stereotyped with the `form` stereotype; and (iv) the navigational relationships among pages is mainly represented using navigated associations stereotyped with the `link`, `forward`, or `submit` stereotype.

The key underlying idea in our approach is to provide input, output and process stereotypes to the action nodes of activity diagrams. Thus, these nodes can be translated into input, output and processing UML WAE classes. In addition, the control flow of activity diagrams is translated into forward/submit messages of sequence diagrams. Finally, decision nodes are translated into `alt` combined fragments where every operand forwards the control to the corresponding page. The concrete translation depends on the target architecture selected. Table 1 provides the translation from activity diagrams into UML WAE diagrams considering a Model 2 architecture.

Table 1. Translation from activity diagrams into UML WAE diagrams. Model 2 architecture

activity diagram element	UML WAE class diagram element	UML WAE sequence diagram element
input action	`client page` + form	instances of both classes
process action	operation in facade + command + dependence from command to facade	instances of command and facade classes
output action	`server page` + built `client page`	instances of both classes
decision node	-	`alt` combined fragment in the controller instance
flow from input to process	`submit` relation from form to controller + transfer with dependences	message `submit` from instance of form to controller + message execute from controller to command + execute operation from command to facade
flow from process to process	`forward` relation from controller to itself + transfer with dependences	message `forward` from controller to itself + message `execute` from controller to command + execute operation from command to facade
flow from process to output	`forward` relation from controller to output `server page` + transfer with dependences	message `forward` from controller to instance of output `server page` + message `build` from output `server page` to built `client page`
flow from process|output to input	`forward` relation from controller to input `client page`	message `forward` from controller to input `client page` that contains `form`
flow from action to decision node	-	interaction begins inside combined fragment
flow from decision to input	`forward` relation from controller to input `server page` that contains form	guard condition + message `forward` from controller to input `server page` that contains `form`
flow from decision to process	`forward` relation from controller to itself	guard condition + message `forward` from controller to itself + message `execute` from controller to command + execute operation from command to facade
flow from decision to output	`forward` relation from controller to output `server page`	guard condition + message `forward` from controller to output `server page` + message `build` from output `server page` to built `client page`
initial/final node	-	interaction starts/ends

The translation depicted in Table 1 supposes the presence of an *application controller* [1] plus a *command* [8], a *facade* [8] (of *application services* [1]) and *transfers* [1] patterns. Of course, the inner structure of transfer classes should be derived while taking into account the data tier of the application. Note that this translation is focused on the presentation

tier and its interaction with the business logic tier. Thus, a detailed description of the business tier is not provided, and almost no information is provided about integration and persistence tiers. The detailed design of all these tiers should be derived from the domain and use case models of the application. The lack of detailed design of the business tier is consistent with activity diagrams that do not include specification of computational behaviours. That is the reason our approach chooses *actions* instead of *call actions*, and specifically, *call operation actions* [23]. According to this interpretation:

- Input actions are translated into input forms attached to client pages.
- Process actions are translated into commands invoked by the controller [8]. In turn, these commands invoke the computational behaviour of the application, hidden behind a facade [8].
- Output actions are translated into output server pages that generate their corresponding client pages.
- Data flow is represented by transfer objects. These transfers flow from input forms to commands, and from commands to output server pages.
- Decision nodes are translated into alternative combined fragments of sequence diagrams. These fragments represent the control flow encoded in the table that guides the controller.
- Control flow is translated into navigational relationships in the class diagrams, and into messages in the interaction diagram.

A translation for Model 1 architecture is simpler than this translation. Regarding multi-tier architecture, because almost no information is provided about business logic or the persistence tier, its translation does not differ much from the Model 2 translation presented in this paper.

3. Example

As was previously mentioned, our research group was entrusted with the deployment of the UCM Virtual Campus [17]. The UCM Virtual Campus provides several services related to teaching, learning and research activities. One of these services is the use case change password, which allows the changing of any user's password. Figure 1 provides an activity diagram describing the workflow performed during the change of a password. This workflow is enriched with information about the type of action node: input, output and process node.

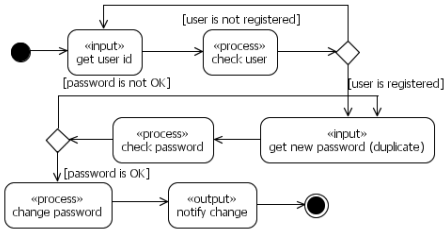


Figure 1. Enriched activity diagram for the change password use case.

Figure 2 depicts a Model 2 architecture class diagram derived from the activity diagram of Figure 1 according to the translation depicted in Table 1. Figure 3 depicts the sequence diagram derived from the activity diagram of Figure 1 according to the translation depicted in Table 1. Although it is unusual to see sequence diagrams involving instances of UML WAE classes, they can be valuable tools for describing traces of browsing. For the sake of brevity, in both diagrams, transfer objects have been omitted.

Note how according to the translation described in Table 1:

- The input action `get user id` in Figure 1 has been translated: (i) into the client page `getUserId` and the form `getUserIdForm` in Figure 2; and (ii) into the instances of these classes in Figure 3.

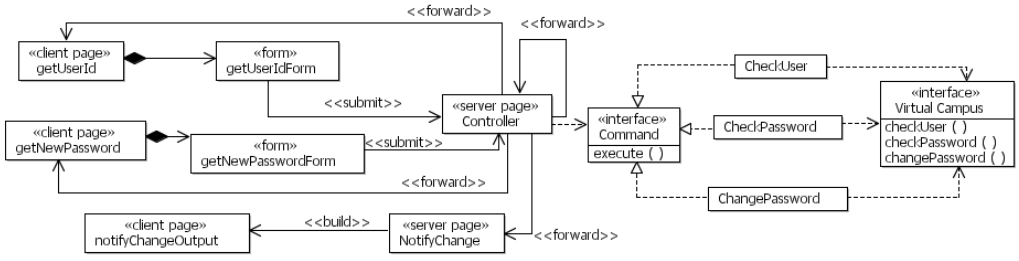


Figure 2. Class diagram for the change password use case. Model 2 architecture

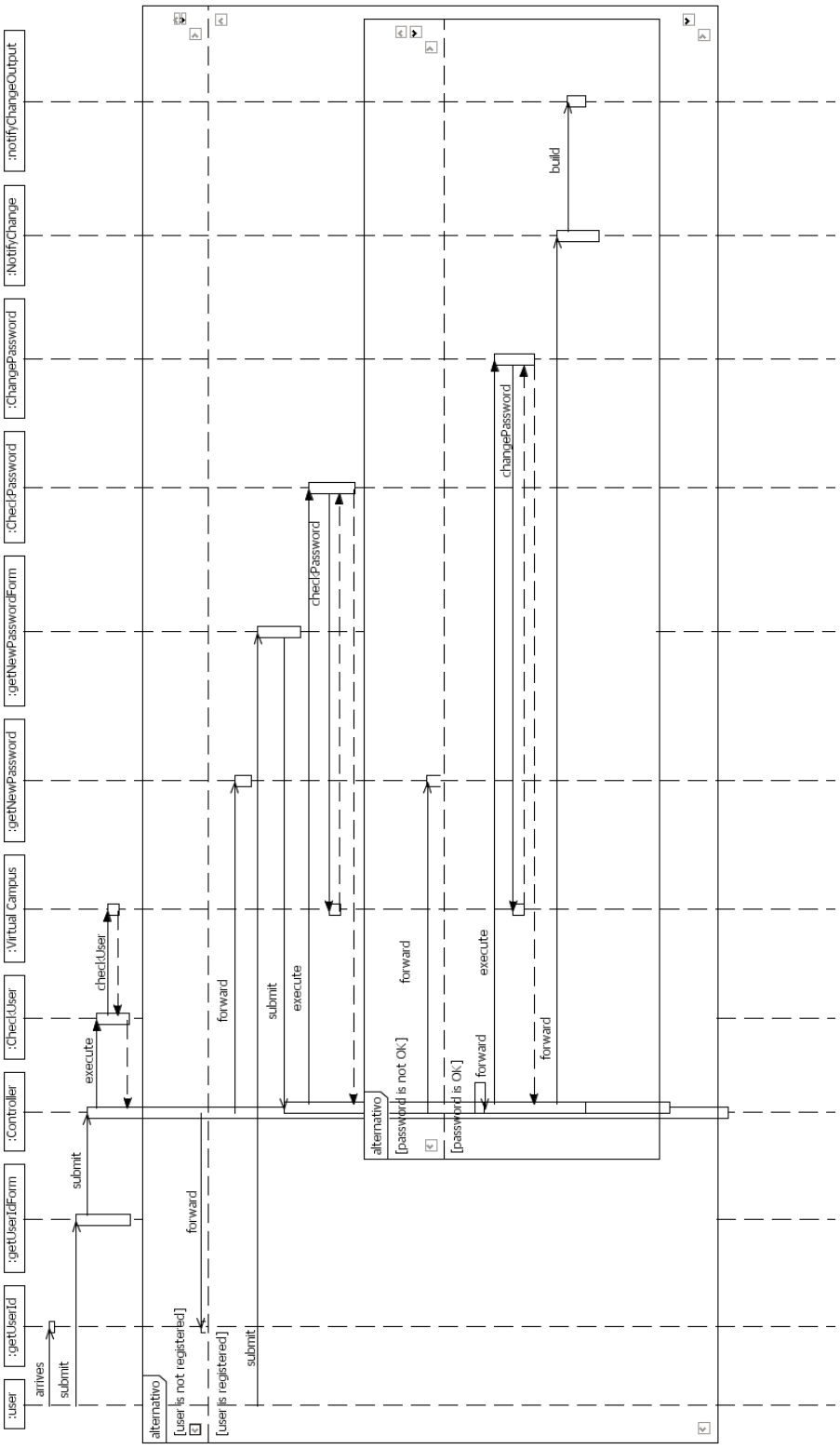


Figure 3. Sequence diagram for use case change password. Model 2 architecture

- The process action `check user` in Figure 1 has been translated: (i) into the command `CheckUser` and the operation `checkUser` belonging to the facade `VirtualCampus` in Figure 2; and (ii) into the instances of these classes in Figure 3.
- The output action `notify change` in Figure 1 has been translated: (i) into the server page `NotifyChange` and the client page `notifyChangeOutput` in Figure 2; and (ii) into the instances of these classes in Figure 3.
- The data flow represented by transfer objects has been left out in the diagrams depicted in Figure 2 and Figure 3 for the sake of clarity.
- The decision node that checks whether the user is registered or not in Figure 1 has been translated into the first alternative combined fragment in Figure 3.
- The flows among these actions have been translated into the relations depicted in Figure 2, and the messages invoked in Figure 3.

The diagrams in figures 1, 2 and 3 were developed using a general purpose UML CASE tool (i.e. *IBM Rational Software Architect* [13]). Thus, the modeling and development effort is not tied to a specific CASE tool. In addition, if *Query/View/Transformations* (QVT) [21] transformations are defined between a *profile* [22] that characterizes the stereotypes defined in this paper and a UML WAE profile, a model-driven approach [20] is enabled for the generation of the platform-independent models of the application (like the one depicted in figures 2 and 3). Furthermore, these platform-independent models subsequently can be transformed into platform specific models, and finally into code.

We are aware that not every workflow diagram can fit in this approach, but this is not the aim of our work. Therefore, although workflow diagrams could be focussed only in the business process specification, which in principle could be completely detached from the concrete web application, our work promotes the provision of high-level descriptions of the use cases of a web application by means of workflow diagrams. If the activities considered in these workflow diagrams are tagged according to the stereotypes presented in this paper, then it is possible to obtain a detailed architectural design of the web application. Thus, the translation from high-level to low-level design is facilitated.

4. Conclusions and future work

Nowadays, a great number of design notations use workflow-oriented diagrams to describe the business processes in web applications. These diagrams are used as high-level design diagrams that, interleaved with design diagrams of each notation, describe the high-level design of the web application. As was previously mentioned, these approaches, although very valuable in describing business processes, leave out the detailed architectural design of the application. Therefore, the maintenance and evolution of the application according to users' demands might be compromised.

This paper provides a UML WAE-based interpretation for activity diagrams. This interpretation helps to understand the architectural meaning of activity diagrams and other types of workflow-based notations (e.g. BPMN). In addition, it provides part of the detailed architectural design of the application in terms of UML WAE diagrams, once a concrete architecture is selected. Thus, the understanding and maintainability of the web application is improved, and the use of standard UML CASE tools is enabled. In addition, because workflow diagrams are nearer to the user than class and sequence diagrams, our approach permits a smooth evolution and adaptation to users' demands. Moreover, a model-driven architecture approach is enabled. This approach is used during the design, development and maintenance of the UCM Virtual Campus.

Regarding future work, activity diagrams are complex artefacts. To provide a complete interpretation to this standard is a goal in our future work. Our aim is to include a translation for call operation actions, extending our translation to the business tier. In addition, our aim is to define QVT transformations between a profile that characterizes the stereotypes defined in this paper and a UML WAE profile. Finally, studying the transition from well-known workflow-based design notations (i.e. UWE and WebML) to UML WAE is part of our ongoing work.

5. Acknowledgements

El Ministerio de Educación y Ciencia (TIN2004-08367-C02-02 and TIN2005-08788-C04-01), *La Comunidad Autónoma de Madrid* (4155/2005) and *La Universidad Complutense de Madrid* (*Group* 921340) have supported this work.

6. References

[1] Alur, D., Malks, D., Crupi, J. *Core J2EE Patterns: Best Practices and Design Strategies*, 2nd edition, Prentice-Hall PTR, 2003.

[2] Barna, P., Frasincar, F., and Houben, G.-J. A Workflow-driven Design of Web Information Systems. *International Conference on Web Engineering 2006* ACM Press, New York, NY, 2006, 321-328.

[3] Book, M., Gruhn, V. Modeling Web-Based Dialog Flows for Automatic Dialog Control. *19th IEEE International Conference on Automated Software Engineering*, 2004, IEEE, 100-109.

[4] Brambilla, M., Ceri, S., Fraternali, P., and Manolescu, I. Process Modeling in Web Applications. *ACM Trans. on Soft. Engineering and Methodology* 15, 4 (2006) 360-409.

[5] Brown, S. et al., *Professional JSP*, 2nd edition, Wrox Press, 2001.

[6] Ceri, S., Fraternali, P., and Bongio, A. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks* 33, 1-6, (2000) 137-157.

[7] Conallen, J. Modeling Web Application Architectures with UML. *Comm. of the ACM* 42, 10, (1999) 63-70.

[8] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[9] Gómez, J., Bia, A., Párraga, A. Tool Support for Model-Driven Development of Web *Applications 6th Int. Conf. on Web Information Systems Engineering,* 2005, 721-730.

[10] Gómez, J., Cachero, C., Pastor, O. Conceptual Modeling of Device-Independent Web Applications. *IEEE MultiMedia* 8, 2, (2001) 26-39.

[11] Hennicker, R. and Koch, N. Systematic Design of Web Applications with UML. In K. Siau, T. Halpin (Eds.): *Unified Modeling Language: Systems Analysis, Design and Development Issues*. IDEA Group Publishing, 2001, 1-20.

[12] Houben, G.-J., Frasincar, F., Barna, P., and Vodvjak, R. Modeling User Input and Hypermedia Dynamics in Hera. *International Conference on Web Engineering 2004*. Springer-Verlag, Berlin, 2004, 60-73.

[13] IBM. Rational Software Architect, http://www-306.ibm.com/software/awdtools/architect/swarchitect/

[14] Knapp, A., Koch, N., Zhang, N., G. Modeling the Behaviour of Web Applications with ArgoUWE. *International Conference on Web Engineering 2005* Springer-Verlag, Berlin, 2005, 624-626.

[15] Koch, N., Kraus, A., Cachero, C., and Meliá, S. Integration of Business Processes in Web Application Models. *Journal of Web Engineering* 3, 1, (2004) 22-49.

[16] Meliá, S., Gómez, J. The WebSA Approach: Applying Model Driven Engineering to Web Applications. *Journal of Web Engineering* 5, 2, (2006) 121-149.

[17] Navarro, A., Fernández-Valmayor, A. Conceptualization of Hybrid Websites, *Internet Research* 17,2, (2007) 207-228.

[18] Navarro, A., Fernández-Valmayor, A., Fernández-Manjón, B., Sierra, J.L. Characterizing Navigation Maps for Web Applications with the NMM Approach. *Science of Computer Programming*, 71, 1, (2008) 1-16.

[19] OMG, *Business Process Modeling Notation*, http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf (2006).

[20] OMG, *MDA Guide. Version 1.0.1*, http://www.omg.org/docs/omg/03-06-01.pdf (2003).

[21] OMG, *Meta Object Facility (MOF) 2.0 Query/View/Transformations Specification*, http://www.omg.org/docs/ptc/05-11-01.pdf (2005).

[22] OMG, *Unified Modeling Language: Infrastructure. Version 2.1*, http://www.omg.org/docs/ptc/06-04-03.pdf (2006).

[23] OMG, *Unified Modeling Language: Superstructure. Version 2.1*, http://www.omg.org/docs/ptc/06-04-02.pdf (2006).

[24] Rossi, G., Schmid, H.A., and Lyardet, F. Customizing Business Processes in Web Applications. *E-Commerce and Web Technologies, 4th Int. Conference,* 2003, 359-368.

[25] Schwabe, D., Esmeraldo, L., Rossi, G., and Lyardet, F. Engineering Web Applications for Reuse, *IEEE MultiMedia* 8, 1, (2001), 20-31.

[26] *UCM Virtual Campus*, http://www.ucm.es/campusvirtual

[27] White, S. *Process modeling notations and workflow patterns*, IBM Corporation BPTrends, http://www.bptrends.com/publicationfiles/03-04%20WP%20Notations%20and%20Workflow%20Patterns%20-%20White.pdf (2004).

[28] Winckler, M. and Palanque, P.A. StateWebCharts: A Formal Description Technique Dedicated to Navigation Modeling of Web Applications. *Interactive Systems. Design, Specification, and Verification, 10th International Worksho*p, 2003, 61-76.

A Security Domain Model for Static Analysis and Verification of Software Programs

Alan B. Shaffer
Naval Postgraduate School
Computer Science Dept
Monterey, CA, USA
abshaffe@nps.edu

Abstract- *Unauthorized information flows can result from malicious software exploiting covert channels and overt flaws in access control design. To address this problem, we present a precise, formal definition for information flow that relies on control flow dependency tracing through program execution, and extends Dennings' and follow-on classic work in secure information flow [7][19][27]. We describe a formal security Domain Model (DM) for conducting static analysis of programs to identify illicit information flows, access control flaws and covert channel vulnerabilities. The DM is comprised of an Invariant Model, which defines the generic concepts of program state, information flow, and security policy rules; and an Implementation Model, which specifies the behavior of a target program. The DM is compiled from a representation of the program, written in a domain-specific Implementation Modeling Language (IML), and a specification of the security policy written in Alloy. The Alloy Analyzer tool is used to perform static analysis of the DM to automatically detect potential covert channel vulnerabilities and security policy violations in the target program.*

I. INTRODUCTION

Identification of exploitable covert channel vulnerabilities is vital in the development of systems intended to enforce mandatory access control policies, and in fact is required for the successful evaluation of such systems at the highest levels of assurance [3][18]. This paper presents a precise, formal definition for various types of covert channels, which depends upon a representation of control flow dependencies, thus extending classic work in this area [7][19][27]. A security domain model is described for formally representing different types of covert channels, and for conducting static analysis[1] of certain program implementations. This model employs dynamic slicing techniques to analyze programs for the existence of access control flaws, where appropriate.

Widely accepted evaluation standards [3][4][18] require that high assurance secure systems be designed, developed, verified and tested using rigorous processes and formal methods. This evaluation process must include demonstration of correct correspondence between system representations at various levels of abstraction, e.g., security policy objectives, security specifications, and program implementation. The Common Criteria for Information Technology Security Evaluation requires that systems at EAL-5 or higher[2] undergo covert channel analysis to ensure that the system is capable of enforcing its security policy in terms of covert as well as overt interactions [3].

Formal security models are often based on concepts of program secure state and state transitions. High assurance evaluation standards [3][4] require a formal verification that the state transitions resulting from program execution preserve the security properties defined by a policy. Our approach analyzes programs for preservation of security properties through state transitions, and advances the concepts of secure information flow in classic work by Denning and others [7][27], by describing automated techniques for information flow static analysis. Previous work in developing our approach has demonstrated the ability to detect illicit information flow security violations [22], and covert channel and overt flaw vulnerabilities based on control flow dependency analysis [23].

The *Implementation Modeling Language* (IML), the first novel element in this approach, is a language that supports basic information processing via assignment statements, conditional and loop statements, read/write statements, file random access, and access to a system clock. Program implementations represented in IML are called *base programs*, and they provide a standardized notation for conducting static analysis of target programs for adherence to a security policy.

The second novel element in this work is the definition of a security *Domain Model* (DM), represented as an Alloy [1][11] specification. The DM provides a framework for specifying program state and state transitions, as well as security-related concepts such as security policy, information flow, access control, and covert channel vulnerabilities. The DM is comprised of an *Invariant Model*, which defines the generic concepts of program state, information flow, and security policy; and an *Implementation Model*, which specifies the behavior of the base program. A specialized *DM-Compiler* was developed to translate a base program in IML into an Implementation Model, and to integrate it with the Invariant Model to form a complete DM specification. The DM is verified using the Alloy Analyzer, which

[1] In this context, *static analysis* refers to analysis of program code without actual program execution.

[2] EAL-7 is the highest Common Criteria evaluation assurance level.

identifies execution paths where the security policy rules are violated.

Whereas many previous security models capture information flow between *objects* and *subjects*, the DM does not explicitly define an object, but implements this concept through variables. An access table records sensitivity labels for program variables as a means of tracking information flow across state transitions. These labels indicate the sensitivity of data stored within a variable, and may change over time as data flows through the system.

The DM captures the concept of information flows with respect to a system subject for input to and output from an external device or random access file. The subject is essentially the executor of the statement, and has a defined access label. The policy rules define legal information flows based on the relationship between the subject label, and that of the I/O source/destination variable, e.g., in a Write_dev operation, a subject label must dominate a source variable label, in order for the variable to be successfully accessed for writing. This requirement might seem counter to the BLP *-property, however in our approach a Write_dev is modeled as a flow from a source variable to a target device, with the latter specified at the level of the subject label.

Section 2 of this paper provides background discussion on covert channels, control flow dependencies, and dynamic program slicing. Section 3 presents an overview of the DM methodology for modeling programs and security policies. Section 4 summarizes our test results with several program examples. Sections 5 and 6 discuss related work, and planned future work in this research.

II. BACKGROUND

We discuss several computer security concepts relevant to this research.

A. Covert Channels

Covert channels use entities other than data objects as a way to transfer information between system subjects, specifically entities not intended for information transfer [12][14]. Such channels allow processes to transfer information in a manner that violates a security policy [8].

An operating system may virtualize a shared physical resource so that each subject, or equivalence class of subjects, perceives that it has exclusive access to the resource. A covert channel can result from the incomplete virtualization of a resource such that some attribute of the resource remains shared, indirectly.

A common taxonomy of covert channels defines them as being either storage or timing channels [20]. For both storage and timing channels the sender and receiver (typically subjects) must have [12]:

1. Indirect access to an attribute of a shared resource, which the sender can modify, and the receiver can view.
2. A means to initiate and synchronize their actions.

In our analysis, we consider that the primary distinction between a covert storage channel and a covert timing channel is the means by which the receiver observes the change in the attribute:

3. Storage – the receiver views an error message, or other information placed in its address space by the system.
4. Timing – the receiver views changes to the relative timing of "legal" events.

The attribute in question forms a *point of interference* [9] between the subjects. To be the basis for an exploitable covert channel, the interference must also be contrary to the computer security policy – i.e., with a mandatory access control (MAC) policy, the sender's security level must be higher than the receiver's level (with respect to confidentiality) [26].

B. Control Flow Dependency Flaws

Covert storage channels based on control flow dependencies often involve the indirect use of internal resources, such as buffers or non-exported files in a program control decision, to pass information from *High* to *Low* [12][14][15]. In addition to this, our approach is capable of detecting overt flaws based on control flow dependencies.

The approach here for discovering flaws based on control dependencies employs a dynamic slicing analysis. To determine the existence of such a dependency within the program, the chain of statements preceding a value assignment is examined with respect to the access labels of the variables in these statements. If the context of a previous statement includes variables that are higher than the destination, then there is an overt flaw.

The code snippet below would not be classified as having a covert channel since internal attributes are not referenced, however it provides an illustration of a control flow dependency that constitutes an overt flaw. In the example, a constant value is written out to a *Low* external device (s3), depending on the *High* value read into variable `v1` (s1).

```
(s1) Read_dev (High, v1);
(s2) if v1 > 0 then
(s3)   Write_dev (Low, 1);
```

The *Low* value assignment depends on a *High* source (v1) in the `if` block (s2), therefore an implicit flow from `v1` to the *Low* device exists [19].

C. Dynamic Slicing

Slicing algorithms are used as a means of tracing data or control dependencies between variables and statements processed during program execution, traditionally for program debugging purposes [13]. Slicing algorithms generate an executable subset of a program, creating a subprogram whose behavior is the same as the original with respect to some variable. They allow one to isolate the dependencies acting upon that variable.

Slicing algorithms are categorized as either dynamic or static, depending on whether they take into account dependencies derived during one particular program execution path (dynamic), or for all possible execution paths (static).

Since slicing techniques have been shown to be useful in tracking data and control dependencies, they can also provide a means of detecting potential overt flaws based on dependencies. The access labels of variables can be used to determine potential security violations, based on the dependencies between these variables. As an example, consider the following code snippet:

```
(s1) if v3 > 17 then
(s2)   v1 := 0;
(s3) else if v4 = 5 then
(s4)     v1 := 1;
(s5)   else v1 := -1;
(s6) v2 := v1;
```

It is clear that `v2` depends on `v1` (s6). Static slicing can show that `v2` has a dependency on both `v3` (s1) and `v4` (s3), since there is a dependency from each of these to `v1`. With dynamic slicing, however, not all execution paths will result in the same control dependencies, e.g., when the conditional expression in (s1) evaluates to true, the final value of `v2` depends on `v3` but not on `v4`, since (s3) is never executed.

III. SECURITY DOMAIN MODEL METHODOLOGY

An overview of the Domain Model (DM) approach to program security verification is depicted in Fig. 1. The DM includes the definition of program state and transitions between states, as well as security rules, specified as Alloy assertions, representing the generic policy a program must abide by. The DM is composed of an invariant and a variable section, derived from the security rules and a target implementation, respectively.

While there are numerous model checker tools currently available, we chose to use the Alloy specification language primarily because of its ability to represent program language abstractions simply and completely. As Jackson [11] points out, referring to his approach as "lightweight formal methods," Alloy models can be easily created and initially tested early in the development process, and then incrementally expanded. He states that the goal of Alloy was to "obtain the benefits of traditional formal methods at lower cost, without requiring a big initial investment," presumably in time and effort [11].

As with traditional model checkers, Alloy deals with finite models, though it handles them very differently. Model checkers typically build Kripke structures to represent the states and transitions of a program execution. Such finite model structures have limits not easily adjusted by the user during analysis. The Alloy Analyzer tool, however, affords the ability to easily increase the depth of analysis for models as they are developed and expanded. For our approach, Alloy and its Analyzer provide a unique, ideally suited tool for creating and analyzing target program abstractions.

In our approach, a *base program* is an abstraction of a target program implementation, and is written using Implementation Modeling Language (IML) notation [23]. The IML defines a simple domain-specific language that captures the basic capabilities and constructs, with respect to

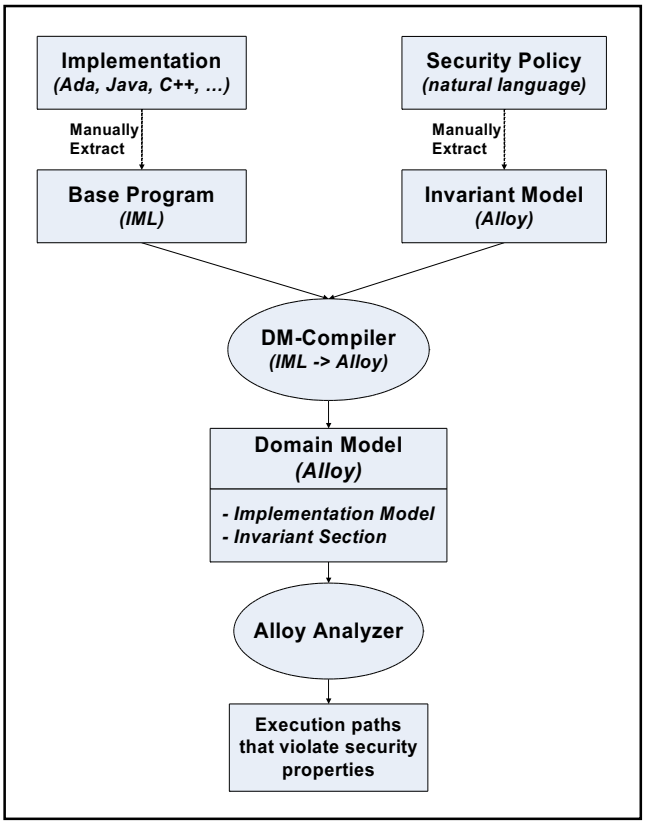


Fig. 1. Domain model approach to system security verification.

security, of high-level programming languages. Our intent is that IML enables the specification of relatively simple programs written in some common programming language, such as Ada, Java, or C++. While future iterations of IML might handle other more advanced language features, e.g., concurrency, inheritance, etc., this initial language description was motivated by a requirement to represent the most essential security information flow properties in target program implementations. This was our goal in describing IML syntax and constructs.

By analyzing a model of the program, rather than actual program code, security verification can focus on elements of information flow analysis, e.g., I/O, access labels, direct file access, and timing (system clock), while ignoring other program details not pertinent to such analysis.

In the current prototype, translation of the base program from an implementation is a manual step. Developing a separate compiler to translate a high-level language program to IML is a difficult task, beyond the scope of this work. The possibility must be considered that overt and covert flow violations existing in the original program implementation may be lost in the IML representation, and for now we depend on the knowledge of the manual translator to avoid this problem.

The Invariant Model includes the definition of security rules, written as Alloy assertions, which must be enforced by the DM security policy. Such policies are typically written in

natural language, and extraction of security rules is a manual step in our approach. As currently implemented, the DM defines security rules associated with the Bell & LaPadula security model [2], i.e., flows from *High* to *Low* secrecy levels are not allowed.

After the base program and Invariant Model with security rules are defined, the DM-Compiler compiles the base program from IML into state transition predicates, written in Alloy notation, creating the DM Implementation Model. The DM-Compiler combines this with the Invariant Model to complete the DM. The approach uses the Alloy Analyzer tool [1] for automated verification of the security rules, defined in the DM as Alloy assertions, to find execution paths within the DM that might violate the security policy or create covert channels. In essence, it creates an interpreter for the specific base program, modeled by the DM. A detailed description of the DM structure can be found at [23].

When analyzing a base program, the Alloy Analyzer performs an exhaustive search of all paths to a defined length (the *scope*, specifying the size of the models considered). In fact, it performs symbolic execution of all base program paths with length up to the given scope limit. In our generated DM, the scope is generated heuristically, based on the total number of statements in the base program. This ensures that all execution paths of that length will be scrutinized. It is assumed that the Alloy *small scope hypothesis*, which states that most flaws in models can be revealed on small instances [11], holds for information flow tracing in our approach.

The Implementation Model of the DM is automatically generated by the DM-Compiler from a base program, and specifies the base program's semantics in terms of statement signatures and state transitions. From the base program, the DM-Compiler generates Value and Variable signatures, representing the number and value of unique constants explicitly present in the base program, and the variables used in the base program, respectively. The DM-Compiler defines an Alloy signature that establishes a less-than relationship between the constant values, enabling comparison of values for equality and inequality in the base program.

The DM-Compiler compiles each base program statement into a separate Alloy signature, based on the type of statement and associated variables and constants used. From these statement signatures, it generates a predicate representing the state transition trace for the base program execution. This predicate captures the semantics of the base program by specifying all possible sequences of statement executions for the program. It also implements dependency tracking within the execution path. A detailed example of this refinement from base program to Alloy signatures and transition predicate is provided at [23].

IV. TESTING AND ANALYSIS OF THE DM

We tested the DM approach using base program examples with illicit information flows, and overt flaw and covert channel vulnerabilities. In each case, a rule for discovering the illicit flow or covert channel is defined as an Alloy assertion, and an example base program is presented to illustrate the error or violation. Each example represents the transmission of one bit of information; more complex examples would involve such concepts as looping, synchronization, etc., to provide the covert channels with a stream of bits.

Our base program examples were evaluated using Alloy Analyzer 4.0. In test runs, the Alloy Analyzer successfully found valid counterexamples for violations of each security rule assertion, i.e., an existing overt flaw or covert channel was detected in each case. The complete Alloy models for these examples can be found at [21].

The "*IllicitFlow*" example [21] demonstrates an illicit information flow based on violation of the BLP simple security policy, i.e., a flow from a *High* object to a *Low* device. The Alloy assertion below defines a security rule for such a policy that examines each execution state, and evaluates to true whenever the state (`s`) is the result of a Write_dev operation to a *Low* device, from a variable whose access label is *Low*. The DM searches for execution paths for which this assertion is not true, i.e., those with a flow that violates the security rule.

```
assert correct_access1{
   all s: State | Property1[s] }

pred Property1 [s: State]{
   let stm = s.stmt | {
      (stm.type = Write_dev and
         stm.subject_label = Low and
         stm.source in Variable)
      => s.access[stm.source] = Low }
}
```

The base program below is an example of a violation of this security assertion. The program first reads a value into variable `x1` at a *High* access level, and then checks the variable's value against a constant. Based on the result of this conditional check, the value in `x1` is either written to a *High* or a *Low* external device.

```
(s1) Read_dev (High, x1);
(s2) if (x1 > 3) then
(s3)   Write_dev (High, x1);
(s4) else Write_dev (Low, x1);
(s5) Stop;
```

The violation occurs when the conditional (s2) evaluates to false, thus the value of `x1` is written to the *Low* device (s4), creating a flow from *High* to *Low*. The Alloy Analyzer detects this situation, and reports a violation of the security assertion through statements (s1)(s2)(s4).

Further examples include "*OvertFlaw*" [21], which illustrates an overt flaw based on a control flow dependency. This example shows an exploitation scenario that culminates with an IML Write_dev operation, where the variables written to the external device have been influenced by values at a higher level than that of the device itself. The approach uses dynamic slicing techniques to discover these flow violations.

The "*StorageChannel*" example [21] describes a classic covert storage channel [16] resulting from access to the direct

file by a *Low* subject (who uses a PutDirectFile operation), after a *High* subject has caused it to be full. The Alloy security assertion defines logic to capture this vulnerability by checking for states where the label of the direct file key slot (keyLabel) is higher than that of the subject (subject_label). The nexus of this covert channel is that *High* can write to the internal resource full (indirectly), and *Low* can observe it.

Our "*TimingChannel*" example [21] describes a covert timing channel that occurs when a *Low* subject twice checks the system clock, between which a *High* subject prevents the *Low* subject from executing through execution of a Read_dev/Write_dev or direct file operation. Thus, when the *Low* subject next runs, it can examine the clock to detect this interference with its access to the CPU; these channels are thus often called CPU channels. The crux of this covert channel is that a *Low* subject, the covert channel receiver, has been allowed to observe (by examining the clock) a change in some internal resource (the CPU busy state), which was indirectly affected by the actions of a *High* subject, the covert channel sender.

V. Related Work

Previous research in modeling secure information flow and access control, and in covert channel analysis is described below. We have extended previous work by integrating a language for formally specifying an implementation with a framework for expressing security policies, particularly with respect to covert channel rules and control dependency flaws.

Classic work on secure information flow [6][7] provides a foundation for this research, including the notion of partial ordering of security classes based on the dominance relationship, the idea of labeling state variables to track such flows, as a way to certify a program.

Other approaches have viewed no difference between classes of covert channels, or between covert and overt flows for that matter. These approaches rely on the concept of noninterference, which states that the actions of one subject can have no effect on the output of a lower subject in a system. Goguen & Meseguer [9] described that security policies can be defined in terms of only noninterference assertions, rather than by the combination of access control and covert channel restrictions. Their ideas were further expanded in [10].

Volpano et al [27] furthered the language-based flow analysis work by defining a linguistic type system for secure flow, and rigorously proving the soundness of the core language with respect to noninterference. Well-typed programs are then guaranteed to be noninterfering – and thus secure by this definition – which was the basis for much related research, summarized by Sabelfeld & Myers in their survey on language-based information flow systems [19].

Other work in using sound type systems for secure information flow has focused on type inference, in which the flow of information is automatically determined based on semantic analysis [5][24]. Eventually, Smith & Thober [25] enhanced the linguistic model of secure information flow such that sensitivity labels need be assigned only at I/O boundaries, while the labels of variables and constants, as well as data information flow through a program's execution, are automatically derived relative to the I/O (device) labels.

Our DM-Compiler similarly tracks the flow of data based on the input device label with no requirement to annotate the code in any other way. Our work differs from the linguistic type system approach in that, rather than constructing a type-safe language with which to write secure programs, we apply abstract interpretation to the analysis of programs in order to detect potential problems and otherwise demonstrate their security with respect to select security properties. Our approach is based on exhaustive information flow tracing of all execution paths in a program, to a certain length (determined by the model scope of Alloy). This tracing is applied for both overt and covert channel static analysis, using dynamic slicing techniques where appropriate such that read-up, as well as violations of noninterference, are detected [28]. Additionally, we provide a compiler to generate a formal specification of a program. Although it yet lacks a formal soundness proof, the DM-Compiler enables generation of formal logic that can be automatically analyzed (using the DM) for secure information flows.

VI. Discussion and Future Work

This paper has provided a survey of ongoing research to develop a formal security domain model for analyzing programs for information flow vulnerabilities, including exploitable covert channels and overt access control flaws. The approach defines a formal security Domain Model (DM) that facilitates specification of security vulnerabilities, independent of program implementation.

Although encoding and checking program semantics and properties is not in itself revolutionary, we feel that this work is evolutionary in extending previous work in the area of information flow tracking based on a precise, formal definition for overt information flaws and covert channels. Our model provides a means of conducting automated static analysis of a program implementation within a finite scope of execution paths. Flow control dependencies and related overt flaws are analyzed using dynamic slicing techniques. This paper has shown the feasibility of this approach on a specific set of examples, within a finite scope.

The Alloy Analyzer guarantees, by the small scope hypothesis [11], that most program errors should be revealed in relatively small counterexamples. Using the Analyzer to perform static analysis of the DM provides assurance that, within a specified search scope, a counterexample will be found when one exists. This means that false negatives and false positives are eliminated within the defined scope.

Future work will focus on formally proving the DM, and on extending its capabilities. In the former case, formal semantic analysis of the IML and DM-Compiler is needed to ensure that the artifacts of each (e.g., the base program and DM Implementation Model) are accurate refinements of the original target implementation. As pointed out in [19], information flow analysis should take place "as close to the executed code as possible." Analysis of a compiled

abstraction of the execution code creates a requirement for trustworthiness in the compiler, as well as the code itself. In addition to semantic analysis of these DM components, the results of the Alloy Analyzer acting on a compiled DM must be formally proven to be both sound and complete, i.e., that they produce neither false positives nor false negatives, respectively.

Work has begun to implement the notion of a trusted subject into the DM. This class of subject is trusted to circumvent certain access control policy rules, to allow such actions as regrading of objects, e.g., downgrading a *High* labeled object to a *Low* level. This requires defining a separate trusted subject policy within the DM, and the ability for the model to administer multiple policies, i.e., for regular and trusted subjects.

Other planned work includes expansion of the DM to enable support for dynamic security policies [16]. This concept would allow the DM to support multiple polices in existence during program execution, with the ability of a system to adapt different policies based on a dynamically changing security environment [17].

REFERENCES

[1] *The Alloy Analyzer*. (2000). Retrieved March 3, 2008, from the Alloy Analyzer website: http://alloy.mit.edu/.

[2] Bell, D., & LaPadula, L. (1973). Secure Computer Systems: Mathematical Foundations and Model, *MITRE Report*. The MITRE Corp.

[3] *Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model*, version 3.1. Document number CCMB-2006-09-001. September 2006.

[4] *Department of Defense Trusted Computer Security Evaluation Criteria*, DOD 5200.28-STD, National Computer Security Center, December 1985.

[5] Deng, Z., & Smith, G. (2006). Type inference and informative error reporting for secure information flow. *Proceedings of the 44th ACM Southeast Conference* (pp. 543-548). Melbourne, Florida.

[6] Denning, D. (1976). A lattice model of secure information flow. *Communications of the ACM*, 19(5), 236-242. ACM Press.

[7] Denning, D. E., & Denning, P. J. (1977). Certification of programs for secure information flow. *Communications of the ACM*, 20(7), 504-512. ACM Press.

[8] Gligor, V. (1993). *A guide to understanding covert channel analysis of trusted systems*. Technical Rep. NCSC-TG-030, National Computer Security Center, Ft. Meade, MD, USA.

[9] Goguen, J., & Meseguer, J. (1982). Security policies and security models. *Proceedings of the IEEE Symposium on Security and Privacy* (pp. 11-20). IEEE Computer Society Press.

[10] Haigh, J.T., & Young, W.D. (1987). Extending the noninterference version of MLS for SAT. *IEEE Transactions on Software Engineering*, SE-13(2), 141-150.

[11] Jackson, D. (2006). *Software Abstractions: Logic, Language, and Analysis*. Cambridge, MA, USA, and London, England: MIT Press.

[12] Kemmerer, R. (1983). Shared resource matrix methodology: An approach to identifying storage and timing channels. *ACM Transactions on Computer Systems*, 1(3), August 1983. ACM Press.

[13] Korel, B., & Rilling, J. (1997). Dynamic program slicing in understanding of program execution. *Proceedings of the 5th International Workshop on Program Comprehension* (pp. 80-90). Dearborn, MI, USA: IEEE Computer Society.

[14] Lampson, B. W. (1973). A note on the confinement problem. *Communications of the ACM* 16(10), 613-615. ACM Press.

[15] Levin, T., & Clark, P. (2004). A note regarding covert channels. *Proceedings of the 6th Workshop on Education in Computer Security* (pp. 11-15). Monterey, CA, USA.

[16] Levin, T., Irvine, C., & Spyropoulou, E. (2006). *Quality of security service: Adaptive security*. Handbook of Information Security (H. Bidgoli, ed.), vol. 3, pp. 1016–1025, Hoboken, NJ: John Wiley and Sons.

[17] National Security Agency IA Directorate. (2004). *Global Information Grid Information Assurance Reference Capability/Technology Roadmap*, Version 1.0.

[18] National Security Agency. (2007). *U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness*, Version 1.03.

[19] Sabelfeld, A., & Myers. A. (2003). Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1), 5-19. IEEE Press.

[20] Schaefer, M., Gold, B., Linde, R., & Scheid, J. (1977). Program confinement in KVM/370. *Proceedings of the 1977 Annual ACM Conference* (pp. 404-410). ACM Press.

[21] *Security Domain Model Project*. (2008). Retrieved March 5, 2008, from Naval Postgraduate School (NPS) Center for Information Systems Security Studies and Research (CISR) Projects website: http://cisr.nps.edu/projects/sdm.html.

[22] Shaffer, A., Auguston, M., Irvine, C. and Levin, T. (2007). Toward a security domain model for static analysis and verification of information systems. *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling* (pp. 160-171). Montreal, Canada.

[23] Shaffer, A., Auguston, M., Irvine, C., and Levin, T. (2008). *A security domain model to assess software for exploitable covert channels*. Manuscript submitted for publication.

[24] Simonet, V. (2003). Type inference with structural subtyping: A faithful formalization of an efficient constraint solver. *Proceedings of the Asian Symposium on Programming Languages and Systems* (APLAS'03), vol 2895 (pp. 283-302). Beijing, China: Springer-Verlag.

[25] Smith, S., & Thober, M. (2007). Improving usability of information flow security in java. *Proceedings of the 2007 Workshop on Programming Languages and Analysis for Security* (pp. 11-20). ACM Press, New York, NY.

[26] Tsai, C., Gligor, V., & Chandersekaran, C. (1990). On the identification of covert storage channels in secure systems. *IEEE Transactions on Software Engineering*, 16(6), 569-580. IEEE Press.

[27] Volpano, D., Smith, G., & Irvine, C. (1996). A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3), 167-187.

[28] von Oheimb, D. (2004). Information flow control revisited: Noninfluence = noninterference + nonleakage. *Proceedings of the 9th European Symposium on Research Computer Security* (pp. 225-243). Sophia Antipolis, France.

Component based architectures for eXtreme Transaction Processing

Luca Vetti Tagliati
LucaVT@gmail.com
Senior Architect at Lehman Brothers
PhD student at Computer Science and Information Systems
Birkbeck University of London

Abstract

This paper presents a methodology aimed at allowing architects to systematically design component-based architecture particularly suitable for extreme transaction processing (XTP). XTP, as defined in [1], is an emerging approach that is becoming indispensable to enabling mission-critical systems to cope with exceptionally demanding performance dictated by the corresponding business growth. In particular, XTP aims at allowing large-scale mission critical, transactional and distributing systems to effectively satisfy the extremely demanding non-functional requirements, especially in terms of low latency and high throughput. In extremely competitive environments, such as investment banking -where this methodology has been tested- high-performance IT systems can enable the organisation to obtain and to maintain a predominant position in the market, which in turn results in a greater ROI. Although a large body of knowledge related to the strictly-technological domain is available, there is a glaring deficiency in the corresponding methodology space.

1. Introduction

The term XTP has been defined by Gartner ([1]) to indicate a new generation of applications designed to cope with extreme non-functional requirements in terms of low latency and high-throughput originated by genuine business needs. In a number of environments, such as equity front office, the ability of responding to a Request For Quote in sub-seconds determine the possibility of making a deal: empirical studies show that 95% of the time, institutional clients select an offer from among the first three returned. Another example is given by the possibility of exploiting market arbitrages (zero risk trades). Often such possibilities appear for a fraction of a second. Therefore, only systems able to determine these scenarios and react in sub-second are able to allow the company to enhance its income with virtually no risk.

XTP brings to mind the early TP (Transaction Process). However, the one-letter difference of these two terms hides a deep divergence, non only in terms of the NFRs but also in terms of the corresponding architecture. In particular, traditional TP were based on proprietary and expensive hardware, often paired with equally expensive and proprietary software. XTP systems, on the other hand, must be designed to scale horizontally on commodity hardware and software based on industry standards in order to increase the ROI. Although the vast majority of XTP initiatives ([1]) are in their infancy, the market already offers a number of enabling technologies, such us performing ESB, clusters and distributing cache. Regrettably, this rich offering in the technology space does not seem to have been paired with a corresponding methodology evolution and, in fact, the gap in this area is noticeable. The methodology presented in this paper expands on the one introduced by J. Cheesman and J. Daniels ([2]) in order to enhance its suitability for new generation multi-layered architecture and to systematically satisfy XTP requirements.

2. Reference architecture

This paper considers the architecture definition in sync with the one proposed in the Rational Unified Process, where it is defined as the set of significant decisions about the organisation of a software system, the selection of the structural elements and their interfaces of which the system is composed, together with their behaviour as specified in the collaborations among those elements, the composition of these structural and behavioural elements into progressively larger subsystems, and the architectural style that guides this organization, these elements and their interfaces, their collaborations and their composition". [4]. Although at the moment in the IT industry there is not a general agreement about the definition of architecture, the entire IT community agrees with the fact that modern architecture should be layered. The architecture selected in this paper envisages the following layers: Presentation, Business Service, Business Object and Integration (see figure 1). The outermost architectural layer is the **Presentation Layer**, which encapsulates the interfaces with external actors and provides/enforces a number of common services like

internal security, data validation and transformation, etc. This layer is typically further specialised in two sub-layers: the User Interface Layer and the Systems Interface Layer, in order to provide a convenient mean to accommodate different actor specific needs, preserving the same business layer.

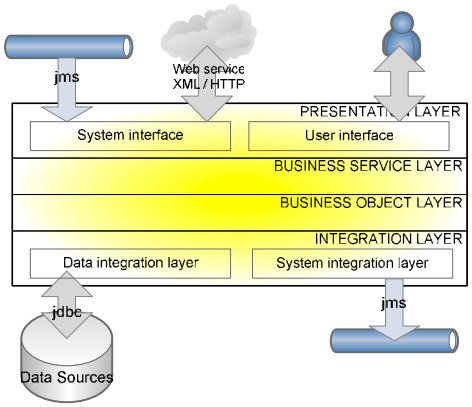


Figure 1. Multi-Layered architecture

The **Business Logic** is the core of the system and this is where the "business rules" are encapsulated. Typically, this layer is split up into two sub-layers: business service and object business layers. It is common to expose the services through a **Business Service Layer (BSL)**. This layer provides the only means of access to the business services of the sub-system. Operations provided by this layer should be derived directly from the steps in the use cases. Each operation that is provided by this layer is a self-contained unit of work. Once a service has been requested it is typically delivered by executing specific sub-services of a number of well-defined objects. The representation of these objects, including their services, is implemented in the **Business Object Layer (BOL)**, which is still part of the business logic. Typical components present in this layer are session EJB for J2EE architecture or traditional bean (often called POJOs) for other frameworks. The BOL consists of core business information, rules, and transformations encapsulated by the sub-system. The BOL components provide core services related to the business domain of the system. Therefore, business services perform their tasks by enhancing services provided by the BOL. Both of these layers are populated by session beans, normally stateless session EJB for J2EE architecture, or POJOs for other architectures. Typical methods that will be present in this BO beans are CRUD services. The **Integration layer** is the final layer is the one used to interact with external data sources and external systems. These can be external databases, other messaging systems, specific gateways, etc. The structure of this layer is equivalent to the one used for the presentation layer: there are a number of standard services that can be extended, via proper plug-ins, in order to enable the integration of specific underlying technologies. In particular, this layer is typically split into the following two sub-layers. **Data Sources**: aimed at integrating databases management systems. The vast majority of existing systems are based on relational databases because these present a higher level of maturity, their capacity and performance are more predictable and reliable. In this case, this layer includes frameworks like TopLink, Kodo, and Hibernate, necessary for mapping classes into tables and vice versa, in other words, adapting the OO paradigm to the relational one. **External Systems**: aimed at integrating with external systems using messages or other specific protocols. In case a MOM is employed, then this layer will include a set of Message Driven Beans. A fundamental rule of this architecture is that the dependencies between components are strictly downwards: a given component can only interact with components in the same layer or with those located in the lower layer.

3. Componentisation

Having defined the reference architecture, including its layers, it is necessary to populate them with well-designed components. However, before proceeding it is important to address the first issue: components located in the different layers of the architecture have to communicate with each other via well-defined interfaces. Considering a simple service like "int = sum(int a, int b)", in this case, the corresponding interface would include three parameters: the two int values in input and the corresponding sum in output.

However, the proposed architecture envisages components that implement coarser-grain services (as per SOA guidelines). Therefore, the basic parameter types are replaced by more complex object graphs, typically called Value Objects (VO) or Data Transfer Objects (DTO), as per the corresponding patterns. These are simple objects whose only goal is to transport data through the different architecture layers: they only expose a collection of getXX/setXX methods. E.g., considering a flight booking service, it would include services like "list<FlightVO> = checkAvailability(AirportVO start, AirportVO dest, ScheduleVO time)", where FlightVO, AirportVO and ScheduleVO are well-defined object graphs. An effective strategy to design these objects consists of starting from the requirements, particularly from the domain object model (DOM), and then clustering this into self-contained and homogeneous graphs to the extent possible. The main criteria for dividing the DOM into sub-domain models consists of applying the O.O. principles, starting from a core business entity

(identified by the stereotype <<core>> in [2]). These elements, the core part of the corresponding object graph, have an independent life in the business and are referred by other dependent entities. Therefore, each graph has to include only highly-cohesive business entities (i.e. classes) which present a low degree of interaction (low level of coupling) with the remaining parts of the system. E.g., in the case of the DOM of an investment bank, it is possible to cluster classes related to market quotes, trade, legal entities, and so on. In an airline ticketing system, it is possible to characterise airport-related information, customer data, aircraft information, itinerary related data and tickets. Once the clustering activity has been carried out, the quality of the division should be verified. A valuable test is to allocate each requested service - *use case or steps of use cases, depending on the use case granularity* - to the group of data that it requires. This procedure consists of balancing the domain object model with the use case model ([3]). This exercise makes it possible to verify basic software engineering laws such as: highly cohesive domains (and therefore conceptual components), low level of coupling between components, minimisation of components communication and logical grouping of services.

When the initial clustering is done, it is typical to have a number of cases where a class belonging to a cluster is associated with another belonging to a different cluster, compromising the low-coupling level and, more importantly, the possibility of implementing self-contained components. These scenarios are quite normal; DOM is a representation of a given business area and therefore it is normal to expect that the different entities that compose it are part of a complex web of links. E.g. although a Trade is a VO, it is quite normal that it is linked to a counterparty, it deals with a well-defined instrument, etc. For this reason, the next step consists of breaking these extra-component relationships: each single object graph must be self contained. A technique used to achieve this is to adopt the referential strategy used by relational databases, i.e. the key export, and to enhance it with some O.O. principle as described in fig. 3. In particular, a relationship between classes can be made implicit by exporting the object id of the referenced object to a simple class called RefXXVO. This can be achieved via a reference class. This offers the advantage of allowing a transparent export of unique codes (TradeVO via its relationship with RefCounterPartyVO imports the sole counterparty code) and the possibility of disassembling and reassembling of complex graphs into/from smaller object graphs. For example, trade data can be split into its parts like: CounterParty, OwnEntity, Instrument, etc.

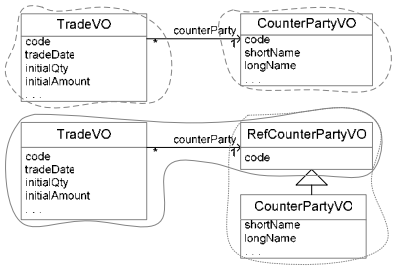


Figure 2. Domain Object Model clustered

Considering a scenario of a user requiring a specific trade, one would expect that he/she would want to view all information. To achieve this, the corresponding BS has to retrieve the different VOs from the various BO component and then to reassemble them to present to the presentation layer a complete and consistent view. The opposite process happens when the BS has to book a new trade. In this case, the initial set of information has to be split into its sub-graphs. The DOM clustering and, consequently, the design of the VOs is a fundamental step used to derive business object components, internal and external interfaces, etc. In fact, once the clustering is concluded, it is possible to assign each cluster/VO graph to a corresponding component at the BO layer. There is a 1 to 1 relation: one value object graph maps to one component. The main role of the BO is to provide a transparent integration to the underlying devices. For example, the most common interface implemented is a CRUD (Create, Read, Update, Delete) aimed at integrating a database, LDAP, etc. Other typical integrations are related to messaging middleware (see fig. 1). The CRUD interface is sufficiently standard: what changes is the referenced VO. They are stateless services that need the required data. For example, the portion of the model depicted in figure 2 would generate two components: TradeBO and CounterPartyBO. Each of these components would expose services like:

- `TradeVO = insert(TradeVO newTrade)`
- `TradeVO = update(TradeVO aTrade)`
- `TradeVO = findById(String tradeCode)`
- `List TradeVO= findByExample(TradeVO aTade)`
- `publish(TradeVO aTrade)`

Due to space constraints, the necessary exceptions have not been included in the method signatures.

The rationale behind the BO tier is to provide a layer of indirection from the underlying devices. In this way the business logic (encapsulated in the BS components) does not need to be aware of the number of data sources, their typology, the vendor, etc. Having also designed the BO layer, the architecture now includes a number of basic blocks able to handle CRUD operations, publishing services, etc., for the

main business entities via a set of precise interfaces based on well-defined VO graphs. At this point it is important to focus the attention on the BS layer components. For this purpose, it is necessary to consider the functional requirements and specifically the use cases. In particular, from the analysis of use case steps, is possible to derive services that, acting on the business objects, allow the component to deliver the required services. For example, considering the Trade, it is possible to identify services like: book a trade, settle a trade, cancel a trade, etc. Therefore, the use case model is the main input artefact used to design the BS layer components, while the DOM is the main one used to design the BO layer (see fig. 3).

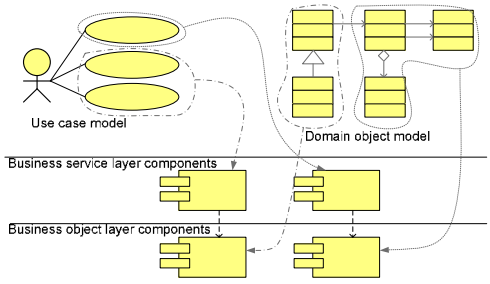


Figure 3. From use cases and DOM to components

Nevertheless, also the BS components have to expose services that act on coarse-grain object graphs and therefore they require VOs as well. Architects often decide to shield the internal business object representation (internal VO graphs) from the exposed one. Therefore, they design a corresponding set of object graphs that the system exposes to its actors. These, as expected, must be able to be transformed into the internal ones, but they do not need to be the same. This allows the reduction of the ripple effect on the internal layers due to change to the external interfaces.During the process of designing the BS components, it is quite common for the architect to review some decisions made during the design of the VOs and BO components. This is necessary in order to increase the level of isolation between components, to improve the performances of some specific services, and so on. This is not a problem since at this stage the architecture is still under design. A completely different scenario would have been generated if changes would have been identified at implementation time. Once the design of the BS and BO components is completed, the architecting of the component of the other layers is straightforward. In particular, the integration layer components are given by the technological decisions (e.g. Hibernate, TopLink, etc), while the presentation layer components are given by the interfaces designed for the UI, to integrate other systems, etc. These are elements of the requirement models that are encapsulated in the same domain object model used to derive all other components.

4. XTP

Once that architecture has been defined and all different components have found their place, including the corresponding interfaces based on VO graphs, it is time to decide which solution to adopt to enable the systems to satisfy extreme non-functional requirements. One of the basic and indispensable principle is to design and implement stateless services (as per SOA guidelines [5]). This is easily achieved in the reference architecture given that all interfaces are based on VOs, which encapsulate the corresponding status. This design enables services that are statefull by requirements to be implemented via a set of stateless services, where the status can be maintained adopting a number of techniques. For example, it is possible to assign this responsibility to the presentation layer (e.g. maintaining the status in the servlet session, or sending it back and forward between clients and services), to the BO layer which resorts to storing the status in the database. Designing stateless services is the necessary but not sufficient condition for allowing the system to scale to the extent that the underlying infrastructure can sustain this, where the infrastructure includes network, messaging systems, database, etc. Currently, vendors offer solutions capable of providing a high level of scalability especially thanks to the combination of modern ESB and cluster deployments based on commodity hardware. This vertical scalability, however, tend to focus mainly on maximizing the throughput which alone is not always sufficient to satisfy most extreme low latency requirements present in specific mission-critical systems. Therefore it is necessary to consider more advanced and sophisticated solutions. Probably the most recurrent one is based on an aggressive adoption of distributed caches in order to implement “all-in-memory” architectures which are supported by the vast availability of non prohibitive memory and, more importantly, 64-bit hardware. One particularly sophisticated strategy envisages the adoption of the cache as “primary data storage”. In this scenario, the data is initially stored in memory and only subsequently and asynchronously persisted into the db. The traditional transaction is therefore split in two asynchronous ones: an initial one necessary for updating the cache and a subsequent completely independent one, to safely store the data into the database. In reality there are several possible subsequent transactions: the initial one can generate the need to update a number of caches (e.g. one for the trade, one for the counter-party, etc.) hence the subsequent database synchronisation can require more

than one independent and asynchronous transaction. For example, the counterparty might not be updated at all, the own-entity can be updated after a few seconds, the trade itself can be persisted after a few minutes, the trade event (e.g. book a trade, settle leg, etc.) can be persisted after a few minutes, and so on. Given the reference architecture, the addition of a distributed cache is straightforward: it is sufficient to plug-in ad-hoc DAO classes (Data Access Object, [6]) whose main responsibility is to handle the integration with distributed caches (see fig. 5). Although in this paper we consider the DAO pattern, this is a simple architectural decisions. The same concept holds true in cases where architects would prefer to adopt other strategies, like EJB entity beans, JDO, etc. All these strategies would generate a completely transparent cache incorporation (except for the performance improvement).

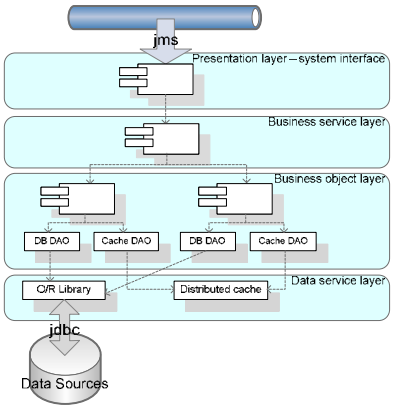


Figure 4. DAOs for the postponed persistence

These new classes are used only by the BO components, hence, the cache incorporation becomes completely transparent to the higher layers of the architecture. In particular, given that each BO component provides for a single VO graph with mainly CRUS services, it follows that each VO graph is encapsulated in a well-defined and separated cache domain. The inclusion of the cache forces BO components to deal with (at least) two kinds of DAOs: the ones designed to integrate the cache and the traditional ones used manage database integration. Therefore, the same service exposed by a BO component (e.g. persist(TradeVO aTrade)) has two implementations: persist in cache (TradeCacheDAO) and persist on the datasource (TradeDBDAO). Hence, the BS components, as expected, continue to use the same interface. The selection of which DAO to use for a given service can be managed explicitly via a corresponding parameter or implicitly via a configurable DAO Factory. Once the distributed cache has been plugged in, the typical service execution includes two main steps:

TX1: the data is persisted into the corresponding caches. Therefore the specific distributed cache domain takes part in the transaction as a transactional resource. For example, if the data is obtained by a message, then the transaction has to include the messaging middleware (message consumption) and the necessary cache domains (VO graph persisted in cache, see fig. 5). This transaction is triggered by a genuine business event, such as a user request, a message delivery, etc.

TX2..n: each not VO graph persisted in the sole cache is read from and then persisted in the database. This transaction is triggered by an internal process (i.e. a scheduler).

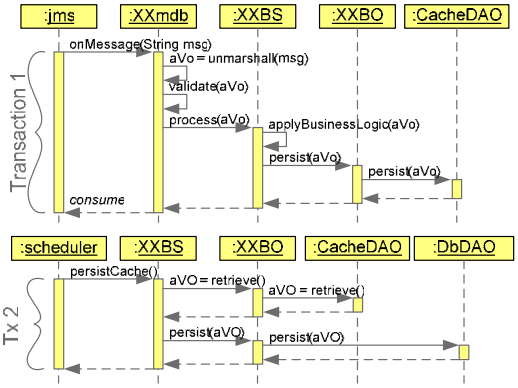


Figure 5. Transactions for the cache and for the db.

As depicted in fig.5, database updates occur in their own transactions that are different from the cache transaction. This architecture based on cache with asynchronous and postponed persistence offers the following advantages: performance improvement especially in terms of low latency -the transactions originated by business events act exclusively in memory and the data persistence is delayed-; enhanced scalability -systems handle more concurrent requests and higher workload can be satisfied by adding nodes in the cluster-; drastic reduction of the database connections – the amount of reading operations is radically reduced and multiple updates often result in a single database transaction -. These advantages, as usual, are coupled with some inevitable disadvantages, like: more sophisticated error handling strategy – database transaction occurs later on and therefore database exceptions must be handled locally without the possibility of rolling back the entire initial transaction -; more complex database integration processes – since the processes to persist data are independent and completely asynchronous, it can happen that the system tries to insert a record in the

table before a referred one is persisted, resulting in an attempt to break integrity referencial constraints-; high-availability requires a number of redundant nodes – since data tends to stay in memory for some time before being persisted in the database, it is necessary to enhance the redundancy in order to avoid data lost -; scalability is not always linear – adding more nodes to the cluster generates an increment on the network communications between cluster nodes to synchronise their status and to generate the requested data redundancy.

5. Case study

This approach has been successfully tested in a large financial organisation with excellent results. In particular, the system proved to be able to handle tens of millions of heavy transactions per day with a very low latency (a second or slightly more during the peak time, 12 GMT when European markets are functioning and the NY market is starting). Furthermore, it proves a more than acceptable approximation of a linear capability to scale on commodity hardware. The following simplified models show some elements of the methodology application.

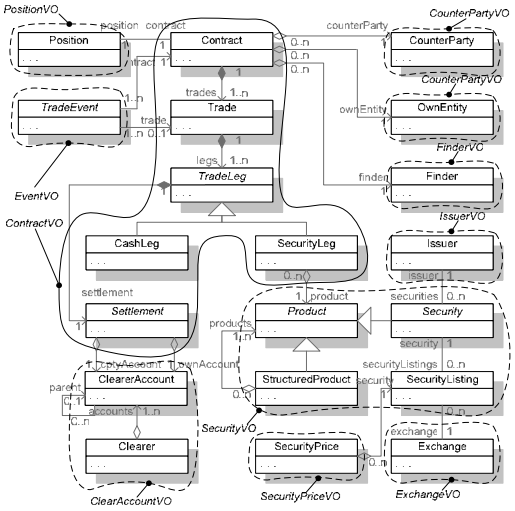


Figure 6. Simplified version of the DOM.

Figure 6 depicts a simplified version of an equity financing DOM. Its analysis might (wrongly) suggests that the right level of granularity for a VO and therefore a BO component is a single class. This is rarely the case. In this example, this is due to the fact that a number of classes have been removed due to space limitations. E.g a CounterParty is linked to its addresses, to a main country of incorporation, the country of residence, etc. It is important to notice that a number of concepts, like counter-party, own entity, security, etc. have only a CacheDAO. This is because the architecture envisages another system whose main responsibility is to manage reference data (or static data) which are transparently distributed to the system client, like the one depicted in figure 7.

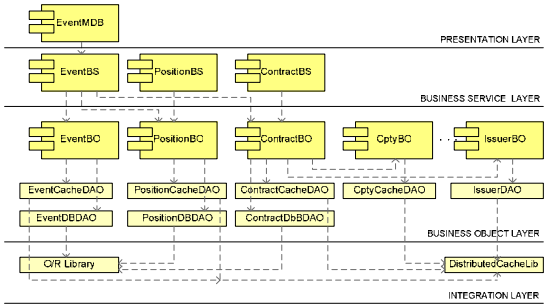


Figure 7. Simplified conceptual architecture view.

6. Conclusion

Although the XTP paradigm is in its infancy status, the vendor market has been proactive in offering a number of enabling technologies. Regrettably, the methodology domain does not seem to have accepted the challenge, and indeed, the gap is obvious. This paper presented a methodology that, starting from the one introduced J. Cheesman and J. Daniels, offers an important enhancement necessary to systematically address multi-layered architectures and more importantly to provide architects with a systematic approach to implement XTP systems able to effectively scale on commodity hardware and standard-based software to increase the ROI. This can be achieved by basing the architecture on the capability to scale of the underlying infrastructure, or, in extreme case, including a high-performing distributed cache. Although this is the main mean of providing very low latency solutions, it also introduces a high degree of sophistication and complexity. Therefore, its usage should be considered only where the business requirements present extreme NFRs.

7. References

[1] M. Pezzini, Milind Govekar, Yefim V. Natis, Donna Scott – "Extreme Transaction Processing: Technologies to Watch" – 13 February 2007

[2] "UML components – A simple process for specifying component-based software" – John Cheesman & John Daniels, Addison-Wesley

[3] [Kruchten: The Rational Unified Process. Booch, Rumbaugh, and Jacobson: The Unified Modelling Language User Guide, Addison-Wesley, 1999]

[4] Frank Armour, Granville Miller – "Advanced Use Case Modelling. Software Systems", Addison Wesley

[5] Yan Qin; Yong Xiang; Meilin Shi – **"**Core-stateless Fair Bandwidth Allocation for Guaranteed Services, Part I: End-to-end Precise Basic Bandwidth Guarantees" **-** Computational Engineering in Systems Applications, IMACS Multiconference on Volume , Issue , 4-6 Oct. 2006 Page(s):1444 – 1448

[6] Sean Sullivan – "Advanced DAO Programming" – IBM – October 2003

An Ontology for Controlled Experiments on Software Engineering

Rogério Eduardo Garcia
Departamento de Matemática, Estatística e Computação
Faculdade de Ciências e Tecnologia – Universidade Estadual Paulista "Júlio de Mesquita Filho"
Rua Roberto Simonsen, 305 – CEP 19060-900 Presidente Prudente – SP, Brazil
rogerio@fct.unesp.br

Erika Nina Höhn, Ellen Francine Barbosa, José Carlos Maldonado
Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo
Av. Trabalhador São-Carlense, 400, C.P. 668
CEP 13560-970 São Carlos – SP, Brazil
{*hohn,francine,jcmaldon*}*@icmc.usp.br*

Abstract

Running multiples experiments in Software Engineering introduces the need of recording data as well as transferring knowledge across them, specially considering that several researchers are involved on replicating experiments. In this work we explore ontologies to support knowledge transfer, helping to elucidate the associated concepts of controlled experiments and their relationships. Based on our expertise on conducting controlled experiments, we have proposed an ontology to experimental studies, named $_{EXPER}Ontology$*. The ontology proposed is intended to be used as a tool for knowledge transfer, assisting researchers, reviewers, and meta-analysts in designing, conducting, and evaluating controlled experiments. In order to validate our ontology we have instantiated it in to a controlled experiment.*

Keywords: Ontology, Controlled Experiments, Experimental Software Engineering, Knowledge Transfer.

1 Introduction

Experimental Software Engineering (ESE) attempts to evaluate and measure the performance of models and techniques in practical contexts, in order to establish a body of knowledge base to support decision-making. Results from a single experiment cannot establish definitive facts about a phenomenon due to variations introduced by different system domains, personal background and experience and cultural environments [5, 9, 10, 12]. Gaining insight into such variations requires running multiple independent studies on a topic, what introduces the need both for recording data and transferring knowledge across multiple studies.

Linkman and Rombach [8] pointed out that experimentation can be used as a tool to support technology transfer. In this sense, lab packages can be viewed as a packing of knowledge about controlled studies. In addition, a lab package must be able to cope with the experiment evolution across multiple studies. A lab package describes an experiment providing materials for its replication, highlights opportunities for variation, and builds a context for combining results of different types of experimental treatments. However, Shull et al. [11] argue that researchers face difficulty understanding and reviewing lab packages. The main difficulties are concerned to the understanding of the concepts underlying the techniques under study, and to mastering of the knowledge involved in running the experiment.

In this scenario, ontologies can be used to enable several researchers to have access to heterogeneous sources of information that are expressed by using diverse vocabularies or inaccessible formats [6]. Based on this, Biolchin et al. [6] have proposed an ontology for ESE in a macro-perspective, showing the concepts of Primary and Secondary Studies on Software Engineering at a high level, aiming at supporting systematic reviews.

In another perspective, we have investigated the establishment of an ontology for ESE, now focusing on controlled experiments – $_{EXPER}Ontology$. Such ontology aims to formally describe the concepts that compose a lab package. The idea is that the ontology can be used as a tool to harmonize concepts and to facilitate lab packages reuse and sharing.

Moreover, since the body of knowledge on testing keeps

The authors would like to thank the Brazilian funding agencies (FAPESP, CAPES, CNPq) and to the QualiPSo Project for their support.

evolving, mainly due to the experimental studies conducted, we intend to merge $_{EXPER}Ontology$ to *OntoTest* [3, 2] – an ontology of software testing we have also defined. At the very end, both ontologies should be used to establish a reference architecture that supports the development of environments/tools to automate software testing activities and related experimental studies. Here we focus on describing $_{EXPER}Ontology$.

The remainder of this paper is organized as follows. In Section 2 we provide a brief overview of controlled experimental studies. In Section 3 we describe the $_{EXPER}Ontology$ and provide an example of its instantiation. Finally, in Section 4 we summarize our contributions and perspectives for further works.

2 Experiments in Software Engineering: Main Concepts

Any experimentation field comprises two types of investigation: Primary and Secondary Studies [6]. Primary studies use specific designs addressed to evaluate the hypothesis formulated by the researcher, to be tested under well-established conditions. Secondary studies intend to produce comparisons between individual investigations selected from a set of primary studies in order to allow generalizing of results. Wohlin et al. [13] have pointed out different sorts of primary studies: Survey, Case Study and Controlled Experiment (see Figure 1). Our ontology focuses on Controlled Experiment. Its main concepts are highlighted throughout Experimentation Process, described next.

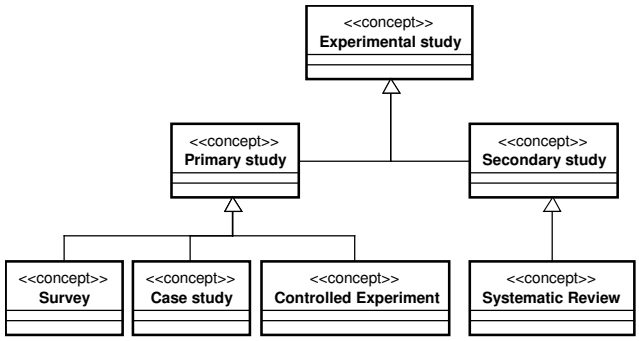


Figure 1. Ontology for Experimental Studies

The Experimentation Process follows a sequence of phases [13]: *Definition*, *Planning*, *Operation*, *Analysis* and *Packaging*. In the *Definition* phase, **hypotheses** are clearly stated and the **experiment goals** are established. Based on the definition, in the *Planning* phase, an **execution plan** must be detailed, defining the **execution environment**, the **subjects** involved and their **profile**, the **dependent** and **independent variables** and their scales. At this stage it is important to discuss the **validity** of the expected results. These two initial phases are iterative, since it is possible to return to a previous phase or redo the current one.

The *Operation* phase is divided into three steps: *Preparation*, *Execution* and *Data Validation*. *Preparation* concerns to preparing the required **material** to run the experiment, such as **data collection forms** and **training materials**. The *Execution* must ensure that the experiment is conducted as planned. Finally, during *Data Validation*, replicators try to check the **data collected** for correctness. These three steps are also iterative. After *Operation*, the data collected is **analyzed** (*Analysis*). The *Packaging* phase is concerned to documentation, including experimental **artifacts**, **procedures** and **results** into a so-called **lab package** for future replications. Amaral et al. [1] suggest that such phase should be conducted in parallel throughout the experimentation process. These concepts highlighted in this section are mapped on the $_{EXPER}Ontology$, presented next.

3 The $_{EXPER}Ontology$

Ontology is a formal explicit specification of a shared conceptualization, that is, an abstract way of perceiving a piece of reality conceived as a set of relevant terms and their relationships, whose structure is constrained by some rules [7]. In short, ontologies are intended for knowledge being used in representation, sharing and management of several different domains.

Based on our expertise on controlled experiments, we have defined $_{EXPER}Ontology$. The idea is to address the main concepts of controlled experiments, from definition to analysis of the results. It is important to notice that we have also considered the experiment evolution, which is stored into the lab package. A researcher runs an experiment to validate some technique, method or tool. In the first running of the experiment a lab package is created, describing the original experiment and providing materials for replication. By analyzing the instantiated concepts from the original lab package, further researches can identify opportunities for variation and build a context for combining results of different types of experimental treatments.

Here, we present the $_{EXPER}Ontology$ in two levels of refinement. The first one refers to the main concepts of a controlled experiment. The second one deals with the refinement of the concepts of validity and lab packages. We have used UML notation to represent the ontology.

3.1 An Ontology for Experiments

From conducting an *original experiment* a *lab package* is generated. A replication uses a lab package from previous experiments as the basis for its motivation as well as for generating a new lab package, as depicted in Figure 2. Both

the original experiment and the replication have to be evaluated regarding to their *validity*. An original experiment is created by a designer, who has his/her profile related to the experiment as a parameter to define a possible threat to validity. In the same sense, a replicator has also his/her profile associated with the replication.

It is important to highlight that both designer and replicator profiles might influence, negatively or positively, the conduction of the experiment/replication. The lack of experience, for instance, is a negative influence since it can be difficult to isolate the factors of risk when defining an experiment. Regarding the replication, the lack of experience can also influence the execution fidelity of the original experiment. On the other hand, the high experience is a positive influence since it minimizes the effort for defining the experiment and helps to analyze the lab package both to identify opportunities and combine results of different experimental treatments. So, the designer and replicator profiles must be taken into account during the analysis of the results as an influence on the experiment validity.

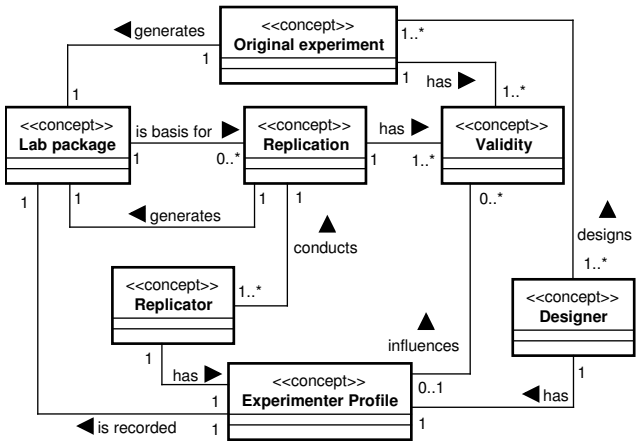


Figure 2. Ontology for Controlled Experiments

The validity evaluation is an issue to be addressed through all phases of the experimentation process. Wohlin et al. [13] pointed out that there are four types of threat to validity: (1) conclusion validity – refers to the relationship between the treatment and outcome; (2) internal validity – refers to the points that assure there is a causal relationship between the factors and the outcome; (3) construct validity – concerns with the relation between theory and observation; and (4) external validity – concerns with generalization. Each type of validity is constrained by threats, as illustrated in Figure 3.

A threat to validity constrains the validity of an original experiment or a replication. However, when there are threats, they are identified in the lab package. The influences to any element that integrate a lab package (or the combination of them) cause a threat to validity. Following, the lab package is defined in terms of concepts.

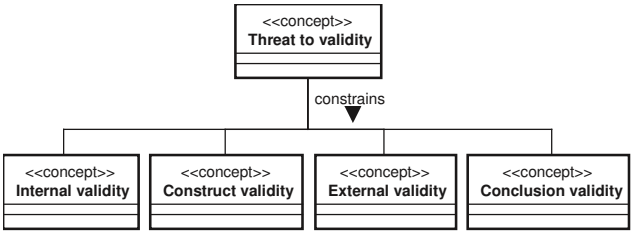


Figure 3. Types of Validity

3.2 An Ontology for Lab Packages

The concepts defined in the ontology for Lab Packages, depicted in Figure 4, are presented throughout the experimentation process, highlighted in the following. At first, the **initial hypothesis** of a controlled experiment is established. It is composed by the **object of study**, in agreement with a **purpose**, under a **quality focus**, and in a specific **context**.

The *Definition* phase is the basis for the *Planning* phase and the **initial hypothesis** generates the **hypotheses formalized**. These hypotheses have **null hypothesis** and the **alternative hypothesis**, as attributes. From the **hypothesis formalized**, the experimenter defines the experiment variables – **dependent** and **independent variables**. During the planning phase s/he also defines the **experiment objects**: **technologies** to be studied (**techniques**, **methods** or **tools**) and **artifacts** (**documents**, **tools** or **forms**) to be used.

Each **subject** has his/her **profile** recorded to characterize his/her background. Capturing the subject background aims at identifying possible influence on results. For instance, previous knowledge about experiment objects or domain application might influence the results obtained. The subjects' profile must be considered to create the **experimental design**, which is built combining experiment objects, independent variables and subjects, in agreement with the hypothesis under investigation. In addition, the subjects' profile must be considered in analysis.

Based on the experimental design, an **execution plan** must be elaborated in order to describe the entire controlled environment to conduct the experiment. Such plan must consider the training activity, which comprises both theoretical and practical approaches for teaching the involved technology. The plan is obtained by establishing the **tasks** to be executed, their sequence and their period. During the execution, each task must have its initial and final time recorded, and differences between **task planned** and **task performed** must be considered as a threat to validity.

The main objective of *Definition* and *Planning* phases

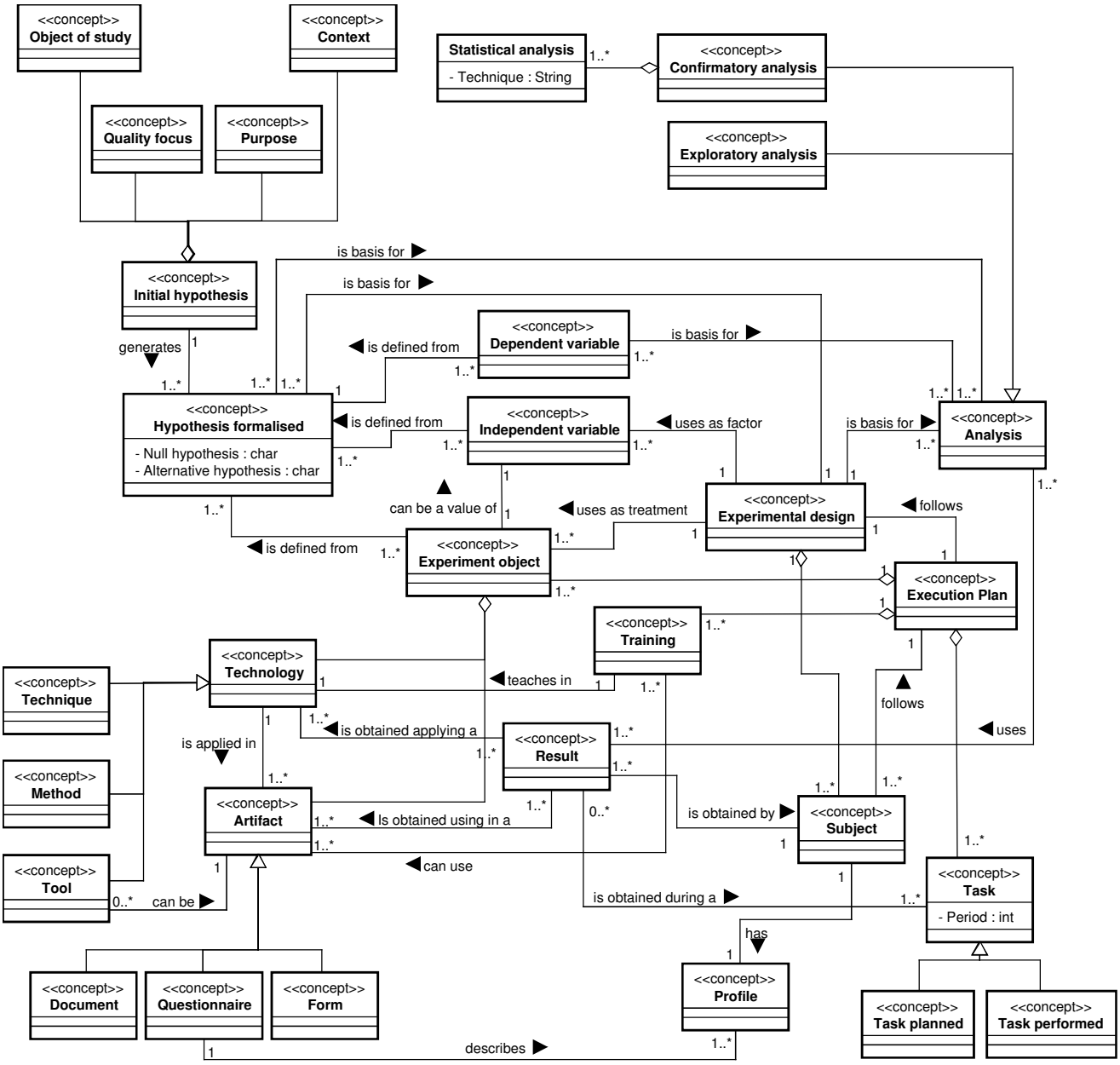


Figure 4. Ontology for Lab Packages

is to establish the experimental design, which must satisfy the requirements to the *Analysis* phase. Such phases culminate in the experimental design and in the execution plan, defining an environment as controlled as possible to test the hypothesis and minimize the threats to validity. Both of them are the core to guide the *operation* phase. The next axiom formally defines the experimental design – the predicate $Design$ associates treatments for each subject:

$$Design(s, SetOfTreat)$$

where s represents subjects and $SetOfTreat$ are values for factors (set of treatment). $Factors$ are defined as:

$$Factor(f_1, \ldots, f_n)$$
$$\forall f \in Factor, \exists\, Treatment(f) = \{v_1, \ldots, v_n\}, n \geq 2$$
$$dom(Treatment) = Artifact \cup Technology$$
$$SetOfTreat = \{(vf_1, \ldots, vf_n)\} | \forall f, vf_n \in Treatment(f_n)$$

where f defines the factor to deal with, and vf_n are values for each f. From the predicate $Design$ it is established the plan for conducting the experiment, which is detailed on a set of tasks:

$$(\forall s, SetOfTreat)(Execution(s, SetOfTreat) \rightarrow Task(ta_1) \wedge \ldots \wedge Task(ta_n))$$

where ta_n defines the tasks. Each task is defined as

$$Task(ta_n) \rightarrow (Training(s, t_r, a, p) \vee Applying(s, t_e, a, p))$$

where the $Training$ predicate indicates that a subject can be trained on a technology (indicated by t_r) using an artifact a, during a period p. In the same sense, the $Applying$ predicate indicates that a subject s applies a technology t_e to an artifact a, during a period p generating data to support testing the hypothesis.

The data set gathered during the execution represents the concept **results** on ontology. The **analysis** of these results is based on hypotheses formalized and on experimental design focusing on dependent variables. **Confirmatory analysis** aims to test the hypotheses formalized and **exploratory analysis** aims to investigate unanticipated relationships.

3.3 Instantiating the $_{EXPER}Ontology$ for a V&V Lab Package

To illustrate the concepts previously discussed, a controlled experiment, originally conducted by Basili and Selby [4], involving V&V (Verification and Validation) techniques is presented. We have chosen such study and used the description found on paper aimed to instantiate the $_{EXPER}Ontology$ using an original experiment created by other researchers, as a validation. The experiment compared the application of three V&V techniques. In order to identify the strengths and weaknesses of each of them. The subjects were selected to be representative of three different levels of computer science expertise: advanced, intermediate, and junior. The experiment had a total of 32 subjects.

The V&V techniques were: (1) code reading; (2) functional testing; and (3) structural testing. The subjects were asked to apply code reading (by stepwise abstraction) to detect discrepancies between the program's abstracted functions and their specifications. The functional testing was performed by applying equivalence partitioning and boundary value analysis to select a set of test case for the program. Then they executed the program on this collection of test cases, and inconsistencies between what the program actually performed and what they thought the specification said were noted. During the structural testing were given the source code for the program, instructions to execute it, and a description of the input format for the program. The subjects were asked to examine the source and generate a set of test cases that cumulatively execute 100% of the program's statements. When the subjects were applying a technique, they generated and executed their own test cases [4].

The experimental design enables the distinction of the V&V techniques while allowing for the effects of the different programs being tested. Three programs were chosen to be representative of several different types of software and provided to the subjects. The three programs selected were a text processor (P1), a numeric abstract data type (P2), and a database maintainer (P3). Table 1 summarizes the Basili and Selby experimental design. Notice that each subject applied the techniques in a pre-defined sequence.

The execution plan was divided into five distinct steps: training, three V&V sessions, and a follow-up session. Elementary exercises followed by a pre-test covering all techniques were administered to all subjects after the training and before the V&V sessions.

Table 1. Experimental Design of the Study Conducted by Basili and Selby[4]

Expertise	Subjects	Code Reading	Functional Testing	Structural Testing
Adv.	S1	P3	P2	P1
Adv.	S2	P2	P1	P3
...				
Interm.	S9	P2	P1	P3
Interm.	S10	P3	P2	P1
...				
Junior	S32	P3	P2	P1

According to the description of Basili and Selby experiment [4] and based on the $_{EXPER}Ontology$, we have instantiated the experimental design concept as following.

$Design(S1, Advanced, Code\ Reading, P3)$
$Design(S1, Advanced, Functional\ Testing, P2)$
$Design(S1, Advanced, Structural\ Testing, P1)$
...
$Design(S9, Intermediate, Code\ Reading, P2)$
$Design(S9, Intermediate, Functional\ Testing, P1)$
$Design(S9, Intermediate, Structural\ Testing, P3)$
...
$Design(S32, Junior, Code\ Reading, P3)$
$Design(S32, Junior, Functional\ Testing, P2)$
$Design(S32, Junior, Structural\ Testing, P1)$

From these instances of $Design$, it is possible to identify the treatments: Advanced, Intermediate or Junior; code reading, functional testing or structural testing; and P1, P2, P3. Such treatments represent values to the $Factor$s: Expertise, Technique and Programs. These treatments and factors are instantiated next.

$Factor = (Expertise, Technique, Program)$
$for\ Expertise,\ Treatment =$
$\{Advanced, Intermediate, Junior\}$
$for\ Technique,\ Treatment =$
$\{Code\ Reading, Functional\ Testing, Structural\ Testing\}$
$for\ Program,\ Treatment = \{P1, P2, P3\}$
$SetOfTreatment = \{(Advanced, Code\ Reading, P3)\}$

Also according to the description, the execution plan concept has been instantiated as:

$Execution() \rightarrow Training(S1, Code\ Reading, --) \wedge$
$\ldots$
$Training(S32, Code\ Reading, --) \wedge$
$Training(S1, Functional\ Testing, --, --) \wedge$
$\ldots$
$Training(S32, Functional\ Testing, --, --) \wedge$
$Training(S1, Structural\ Testing, --, --) \wedge$
$\ldots$
$Training(S32, Structural\ Testing, --, --) \wedge$
$Applying(S1, Code\ Reading, P3, --) \wedge$
$\ldots$
$Applying(S32, Structural\ Testing, P1, --)$

From these predicates we can notice that there are missing values (indicated by $--$). Indeed, the paper describing the experiment do not bring them. Observe that the ontology can also be used as a mechanism to improve the obtained data set from the Lab Package. On the $Training$ predicate there are two missing values: the first one refers to the artifact used in the training activity and the second one refers to the period for the training. On the $Applying$ predicate there is only one missing value, which refers to the period for the application of a technique to a given program.

4 Conclusions and Further Works

In this paper we proposed $_{EXPER}Ontology$, an ontology for Experimental Software Engineering, focusing on controlled experiments. $_{EXPER}Ontology$ has been built based on our knowledge and expertise on conducting controlled experiments, mainly on evaluating V&V techniques. The importance of keeping all the data about controlled experiments on lab packages motivated us to focus on mapping the concepts involved. The idea is that a lab package based on the $_{EXPER}Ontology$ can be used to improve the knowledge sharing among researchers when conducting replications and systematic reviews or meta-analysis.

We have also conducted a preliminary validation of $_{EXPER}Ontology$ in the context of the Basili and Selby [4] experiment. However, it is important to highlight the need of validating and evolving the proposed ontology for other controlled experiments in different domains. Also, we intend to include some established guidelines for experimental software engineering to ensure the validity of the ontology. Further investigation has been planned in this sense. Moreover, we intend to explore how to integrate the $_{EXPER}Ontology$ to $OntoTest$. Such ontology has specified several concepts presented here, as technique and tool, focusing on testing activities. At the very end, both ontologies should be used to establish a reference architecture that supports the development of environments/tools to automate software testing activities and related experimental studies.

References

[1] E. A. G. G. Amaral, W. A. Chapetta, and G. H. Travassos. A process for experiment packaging. VII Workshop on NSF-CNPq Readers Project, April 2003.

[2] E. Barbosa, E. Y. Nakagawa, A. C. Riekstin, and J. Maldonado. Ontology-based development of testing related tools. In *20th Int. Conference on Software Engineering and Knowledge Engineering (SEKE 2008)*, San Francisco, CA, July 2008. Accepted (to appear).

[3] E. F. Barbosa, E. Y. Nakagawa, and J. C. Maldonado. Towards the establishment of an ontology of software testing. In *18th International Conference on Software Engineering and Knowledge Engineering (SEKE 2006)*, pages 522–525, San Francisco, CA, July 2006. Short Paper.

[4] V. R. Basili and R. W. Selby. Comparing the effectiveness of software testing strategies. *IEEE Transactions on Software Engineering*, 13(12):1278–1296, 1987.

[5] V. R. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *IEEE Transaction on Software Engineering*, 25(4):456–473, 1999.

[6] J. C. de Almeida Biolchini, P. G. Mian, A. C. C. Natali, T. U. Conte, and G. H. Travassos. Scientific research ontology to support systematic review in software engineering. *Adv. Eng. Inform.*, 21(2):133–151, 2007.

[7] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

[8] S. Linkman and H. D. Rombach. Experimentation as a Vehicle for Software Technology Transfer - A Family of Software Reading Techniques. *Information and Software Technology*, 39:777–780, 1997.

[9] J. Miller. Can results from software engineering experiments be safely combined? In *6th IEEE International Software Metrics Symposium (IEEE METRICS)*, pages 152–158, Boca Raton, FL, USA, November 1999.

[10] J. Miller. Replicating software engineering experiments: a poisoned chalice or the holy grail. *Information & Software Technology*, 47(4):233–244, 2005.

[11] F. Shull, V. R. Basili, J. Carver, J. C. Maldonado, G. H. Travassos, M. Mendonça, and S. Fabbri. Replicating software engineering experiments: Addressing the tacit knowledge problem. In IEEE Computer Society, editor, *2002 International Symposium on Empirical Software Engineering*, pages 7–16, Nara, Japan, October 2002.

[12] F. Shull, J. Carver, G. H. Travassos, J. C. Maldonado, R. Conradi, and V. R. Basili. *Replicated studies: building a body of knowledge about software reading techniques*, pages 39–84. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2003.

[13] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Boston (USA), December 1999.

Improving Automatic Model Creation using Ontologies

Sven J. Körner
Institute for Programming and Data Structures
University of Karlsruhe
76131 Karlsruhe, Germany
Email: koerner@ipd.uka.de

Tom Gelhausen
Institute for Programming and Data Structures
University of Karlsruhe
76131 Karlsruhe, Germany
Email: gelhausen@ipd.uka.de

***Abstract*—Automatic model creation from textual specifications is a complex task. We show how ontologies can be used to improve the quality of automatically created UML models. An evaluation of a model transformation from a textual specification of the World Chess Federation to UML is used as an example. The resulting UML models are substantially improved.**

I. Introduction

Dealing with natural language textual specifications is a sophisticated task. When developing new software, user requirements are usually written down in textual form. Natural language is used to align the user's understanding of the requirements with the analyst's own viewpoint. Models are being created in later steps. The most popular are UML models.

Finding out what the requirements originator really meant when stating "facts" is complex [1]. Rupp and Goetz show that focusing on the requirements in the first stages of the software development process prevents numerous mistakes [2] and therefore saves effort. We offer an approach and a collection of tools to automatically create models from textual specifications [3].

Today, it is still a manual task to provide enough information for the model creation system so that it is capable of automatically creating a model of the input text. It is common sense that helps humans to extract semantic information when preparing textual paragraphs for machine processing. Douglas Lenat said in 1998 [4]: "If you pluck an isolated sentence [...] it will likely lose some or all of its meaning – i. e., if you show it out of context to someone else, they will likely miss some or all of its intended significance." Computers lack this kind of knowledge.

Applying implicit semantics by using ontologies, we want to overcome this problem and contribute to the solution with outlook to a completely covered process from textual specifications to ready-to-go and ontology-improved UML models.

II. Model Creation

The basis for our research is the SENSE (Software Engineers' Natural language Semantics Encoding) system [3]. It provides the annotation language SALE (SENSE Annotation Language for English) that requires minimal reorganization of the original document.

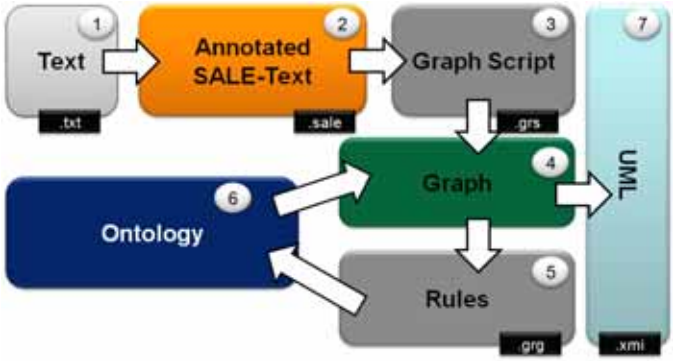


Fig. 1. The Model Creation Process

A. Encoding Semantics

The key concept of SENSE is the encoding of semantics via thematic relations and graph contiguity in omnigraphs, a formal extension of hypergraphs [5]. Thematic relations denote which role a constituent plays in an n-ary relation: The ones used in SALE comprise AG for *agens*, an acting person, ACT for *actus*, the action itself, and PAT for *patiens*, the person or thing being acted on etc. (for a complete list cf. [3]). A short description can be found in Table I.

B. Using Graphs for Representation

The annotation of natural language serves to encode the semantics of a text. The annotated text can be processed via an ordinary ANTLR [6] generated parser thus avoiding error-prone Natural Language Processing (NLP). The SALE compiler creates a script that builds a graph instance in the graph rewrite system GRGEN.NET [7]. In this graph, quested semantics are represented via types and contiguity. The advantages of graph representations are their abstraction from any oddities of the underlying natural language, the direct representation of cyclic content, and that the very same representation can serve for multiple natural languages.

C. Model Creation Process

Creating the UML model takes several steps as can be seen in Figure 1. The process is described in the following.

1) Text: First there is the text of the specification. An example would be:

```
The game of chess is played between two opponents.
The player with the white pieces commences the game.
```

2) Annotation: An editor then annotates the text with semantic information so that the natural language specification is machine-processable. This would look as follows:

```
[ #The game_of_chess|PAT #is played|ACT #between
  *two opponents|AG ].
[ [ #The ^player|POSS #with #the $white
    pieces|HAB ]|AG commences|ACT #the game|PAT ].
```

In SAL$_E$-syntax the hash "#" denotes a comment, the bar "|" delimits the thematic relation, the asterisk "$\star$" shows the multiplicity of attributes and the square brackets "[]" delineate clauses. For more details see [3].

3) Graph Script: The graph (rewrite) script includes the commands to build the graph.

4) Graph: The actual graph is being created from the graph (rewrite) script (.grs).

5) Rules: After creating the initial graph, rules rewrite the graph. We provide a special set of rules which allows us to gather information for later ontology processing.

6) Ontology: After extracting a number of useful thematic relations from the graph, we query the ontology. The idea is to gain as much implicit information as possible from the individual concepts. This information can then be used to improve model creation.

7) UML: We also use graph-rules to transform the already existing graph into UML. The back-end of the SENSE/SAL$_E$ system is meanwhile capable of exporting truly UML compliant XMI documents [8]. This can then be visualized in any UML application, such as Altova UModel 2008.

III. Ontology-Based Model Improvement

UML diagrams not only consist of class diagrams, but also state-charts, sequence-diagrams and so forth. Every thematic relation can be matched to several UML model types (see Table I). This chapter shows that the correct UML representations can be chosen by using ontologies.

As for processing natural language during the UML model creation process, there are two possibilities where ontology-reasoning takes place: during the initial annotation phase (pre-processing), and after having created the model (post-processing). In this paper we focus on the latter one. The model is already created and loaded into the graph as shown in section II-C. In the next phase we extract the concepts we want to process in the ontology directly from the graph. This is done via graph-rules as depicted in Figure 1. Once the necessary ontology knowledge from the UML concepts has been gained, we send the newly gathered information back into the graph. A consolidated UML model is created.

A. Introducing RCyc

We chose the ResearchCyc (RCyc) ontology since it offers a very extensive coverage of real life concepts [9]. WordNet in comparison is optimized for lexical categorization [10] while MIT's ConceptNet [11] differs from the popular concepts mentioned [12].

TABLE I
SEMANTIC ROLES MATCHED TO UML (EXAMPLE)

Thematic Relation	Explanation	UML model element
AG	The acting person or thing executing the action.	Class, Role, or Instance
ACT	The action, executed by person or thing	Method, State (-Transition) or Relation
PAT	Person or thing affected by the action or on which action is being performed	Class, Role, or Instance
HAB	Possession or belonging; person or thing being received or passed on by person or thing	Class, Role, or Instance
...	empty	...

The most important RCyc-predicates for our task are `#$isa` and `#$genls`. The first one describes that one item is an instance of some collection, the second one that one collection is a sub collection of another one. (RCyc-)Facts about concepts are asserted using certain CycL sentences [9].

B. Support Model Creation

We extract lists of thematic relations within interconnected sentences from the graph. The graph contains the text with its phrases, words and thematic relations. It is processed one paragraph at a time. The ontology processes each thematic relation individually (see chess example in III-E).

We start with the AG (Agens) and ACT (actus) thematic relations as shown in Table I. Depending on context and meaning of each word, only certain UML concepts of these n-ary relations are suitable and sensible. Having the phrase "user A uses an interface B in the application" implies a relation between the two. It is not appropriate to model the verb "use" as method in a UML class diagram. It is vital to realize what is reasonable to model and how. Feeding back the gathered information into the graph generation system is part of the approach (see Figure 1). The rewritten graph can then emit a XMI file which is transformed into the improved UML model.

C. Thematic Relation AG (Agens)

Looking at the AG thematic relation of a given phrase, there is a chance that some of these roles are interrelated (e. g. "car" and "vehicle"), conceptually related (e. g. "car" and "highway") or maybe even mean the same (e. g. "car" and "automobile"). Words are often replaced with their synonyms leading to ambiguities. One could say that a good specification should have as few as possible of these, unfortunately, reality proves us wrong [1]. Today's available disambiguation features

cannot repel all problems which might arise when the system cannot recognize all or important polysemy.

Depending on the number of AGs in play, there might be a set of AGs which could be combined. For example we have the AG "User" and another AG "Operator" who work on a "system" described in a specification. Realizing that there is a certain similarity in the two terms should lead to the conclusion that there also is a certain connection between the two. If there are two AGs which happen to have the same generalization, or one of them is an instance of the other, it might also be modeled as instance of a certain class and not as an additional class in the UML model.

Let's assume the graph contains three different AGs A, B, and C. To make sure these items are not equal or at least related in any kind, we compare them pairwise and check their similarity. That means checking if one object is an instance of another (in CycL: `#$isA`) or belongs to the same subset (in CycL: `#$genls`) by querying the ontology. Take the words "player" and "opponents" for example. Players who play against each other are opponents. Thus every player is an opponent. A relation `#$isA` between the "player" and "opponent" class in the UML model is inserted. The methods and attributes of both UML classes are combined.

Querying RCyc about "player" and "opponents" leads to hundreds of facts and connections. A excerpt is listed below.

```
predicate opponents:
isa:  IrreflexiveBinaryPredicate
      SymmetricBinaryPredicate
      CoexistingObjectsPredicate
(argIsa opponents 1 Agent-Generic)
(argIsa opponents 2 Agent-Generic)
(opponents AGENT1 AGENT2)

Collection Player:
isa:  Agent-Generic
```

The above listing shows that "opponents" is a `predicate`. The syntax `(opponents AGENT1 AGENT2)` means that `AGENT1` and `AGENT2` are opponents of each other. Also the arguments are of type `Agent-Generic` as can be seen in `(argIsa opponents 1/2 Agent-Generic)`. Querying not only for "opponent", but also for "player" leads to the RCyc output that a "player" is also an `Agent-Generic` type, just as opponents are. The conclusion is that both words have a relation. This is represented in a UML class-relation and inserted into the graph.

D. Thematic Relation ACT (actus)

The ACT thematic relation is usually denoted by a verb. A verb indicates an occurrence (sparkle, blink), a state of being (exist, is there), or an action (run, cook). Therefore ACT is translated into a state (-transition), relation or a method in UML (see Figure 2). State (-transitions) are instantaneous, relations do not have a fixed duration and are wholly present at time whereas methods are extended in time but are not wholly present.

1) ACT as State (-Transition): When a word such as "win" or "checkmate" is being looked at, it is quite obvious for humans, that it is more likely to be a state (-transition) than a relation (shown in Figure 2 as "TransACT"). Winning a game takes place after all the premises for a victory have been fulfilled. Let's assume State A is the state depicting that the game is still on and one has not won yet. State B shall be the state that marks the end of winning the game. As soon as the game is won, this type of ACT "jumps" from State A to B. It is not a process which takes a discrete period of time to happen. This sort of ACT is represented as state (-transition). Querying RCyc about "win" gives the output below.

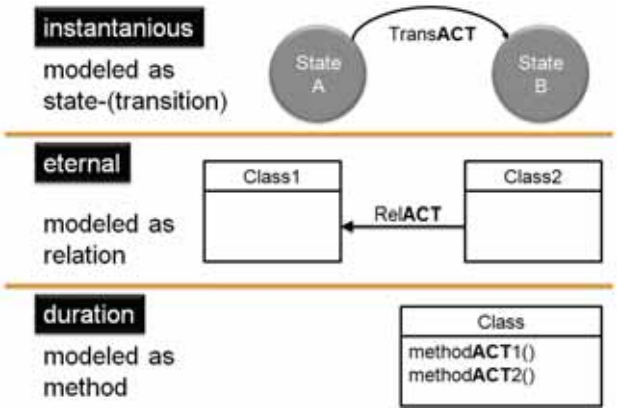


Fig. 2. Classification of Thematic Relation ACT (actus)

```
Collection: Winning
isa:    ConflictEventStatus
        AtemporalNecessarilyEssential
          CollectionType
        AtemporalThing
genls:  AtemporalThing
```

In RCyc the verb "win" is recognized as the collection `Winning`. The digest of the RCyc-facts shows that it is atemporal (RCyc: `AtemporalThing`). It is also generalized by `AtemporalThing`. This means it is a specialization of the collection of all things that are "timeless" in the sense of having no "location" in time. It makes no sense to ask of an atemporal thing, e. g. "When did it begin (or cease) to exist?" Examples of atemporal things include sets, collections, numbers, vectors, and certain "abstract structures" (such as the structure of a partial ordering).

2) ACT as Relation: What if the ACT is instead a stative verb? Than it should not be translated into a state (-transition) in UML, but rather a relation between two classes or their corresponding instances (shown in Figure 2 as "RelACT"). An example would be the word "use" which describes that something uses something else. This denotes a relation. Querying RCyc about "use" leads to the output below.

```
Predicate: usesIn
isa:    TernaryPredicate
arg1Isa:Agent-PartiallyTangible
arg2Isa:PartiallyTangible
arg3Isa:Action
(argIsa usesIn 1 Agent-PartiallyTangible)
(argIsa usesIn 2 PartiallyTangible)
(argIsa usesIn 3 Action)
```

RCyc understands "use" as the predicate `usesIn`. Predicates are represented as UML relations. We further learn that `usesIn` is a `TernaryPredicate` and what kind of input types it takes. The RCyc output explains that the `agent ARG1` uses the `object ARG2` to perform the `action ARG3`.

3) ACT as Method: If ACT represents an action that has a duration or is temporally extended it can be considered as action which leads to a UML representation as method (in a class). Examples would be words like "move", "run", "calculate", etc.

```
Collection: CausingAnotherObjects
                TranslationalMotion
isa:  EventOrRoleConcept
      FirstOrderCollection
genls:ActionOnObject
      Movement-TranslationEvent
      TemporalThing
      TemporallyExistingThing
      TemporallyExtendedThing

Collection: Movement-TranslationEvent
isa:  EventOrRoleConcept
      FirstOrderCollection
genls:TemporalThing
      TemporallyExistingThing
      TemporallyExtendedThing
```

In RCyc those types of ACT can be deduced to `TemporallyExtendedThing`. The output above is from the word "move" which RCyc translates into the collection `CausingAnotherObjectsTranslationalMotion` and also `Movement-TranslationEvent`. Both are being generalized (RCyc: `genls`) by `Temporally-ExtendedThing`. This is the collection of all things that are "extended in time", as opposed to being "wholly present at a time". For example, an event is a temporally-extended thing, as it is extended in time; it is not wholly present at any interval that is properly subsumed by its temporal extent. This is similar for a time-interval, such as a particular calendar year where e. g. 1999 is not present in 2001 anymore. Conversely, a person is not a temporally-extended thing, as she or he exists at different times and is wholly present at each such time. `TemporallyExtendedThing` is marked as UML method.

E. Chess Example

We chose article 1 from the section "Basic rules of play" of the FIDE laws of chess [13]. An excerpt reads the following:

```
The game of chess is played between two opponents
who move their pieces alternately on a square
board called a 'chessboard'.
The player with the white pieces commences the game.
```

When annotating the first sentence we get AG "opponents" with a multiplicity of two and the ACTs "play", "move" and "called". Querying RCyc, "play" is denoted as relation, "move" as method and "called" as another relation. The ACT "called" belongs to "square board" with its thematic relation PAT (patiens) and is therefore not considered in this example. The initial UML model is illustrated in Figure 3.

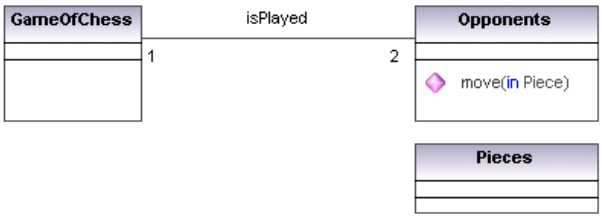


Fig. 3. UML after 1st sentence

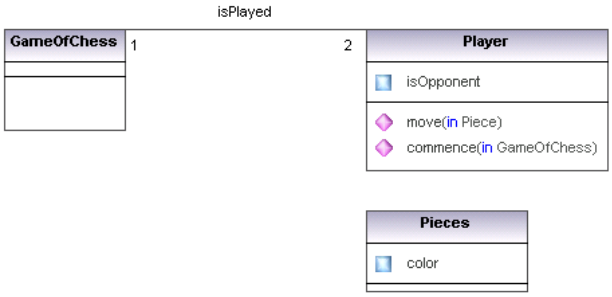


Fig. 4. UML after 2nd sentence

The second sentence introduces AG "player" and ACT "commences". "White" is an attribute of the "pieces". It is also a "color" (`isa`) and therefore modeled as the attribute "color". Querying RCyc about the "player" shows that it is of type `Agent-Generic` and that it is used as an argument for the `(opponents AGENT1 AGENT2)` relation. Thus the AG "player" is similar to "opponents". The user needs to decide which concept should be replaced by the other. We decided for "player". Therefore the UML class "opponent" is not created and "player" receives the attribute "isOpponent" and the methods of "opponent" as well (see Figure 4). The ACT "commences" returns the RCyc result of being a `genls` of `TemporallyExistingThing` and therefore modeled as method.

As can be seen, it is easy for humans to spot that "The player" in the second sentence is also a part of the "two opponents". The computer cannot make that distinction without ontology knowledge. Thus when using the ontology, the models tend to be far more human readable and less complex since the correct type of model is created.

This is true for model creation as far as the ontology has the necessary knowledge. There are some drawbacks which are enumerated in the following.

IV. Evaluation

Having the machine translate the specification without enriched ontology knowledge, UML diagrams grow unnecessarily large. Already small specifications appear far more complex than they actually are. For example, all AG concepts could be treated independently instead of potentially grouped. Additional to that, they would be modeled as class, role and instance at the same time. Also modeling all ACT thematic relations as method, state(-transition), and relation while not

being able to decide which UML type to choose, makes the UML grow large. The automatically created models become mazy and hard to read. The model creation might end up with thousands of classes when hundreds of classes – each including the right modeling – would do. The usability and complexity are not acceptable. Therefore automatic model creation is much more applicable when ontology knowledge is integrated.

Making sure that only one UML interpretation of a certain AG and ACT is created, the complexity of the automatic models is decreased by 2 out of 3 each for AG and ACT. That is 66% less unnecessary and useless concepts.

The hit rate of AGs and ACTs queried in RCyc is acceptable. However, when working with the FIDE chess specifications, the coverage and reasoning of the ontology knowledge was not always adequate. For example the ACT "occupied" is discovered as a `TemporallyExtendedThing` in RCyc which makes it a UML method in our approach. It would fit much better into the category of state(-transitions) since a chess square is either (instantaneously) occupied or not. It could also be modeled as relation between a piece and a square to model which piece occupies which square.

Liu et al. also show that common sense can be regarded as a myriad of simple facts [14]. The overall obstacle when working with ontologies is the sparse coverage of special subjects and the resulting unreliable inferences. As any ontology, RCyc does have its blind and weak spots though it is still the most extensive collection of reasonalbe knowledge we have discovered so far.

We did not make any contribution to the ontology to provide a feeling of what would be possible by just employing the ontology rather then entering domain knowledge beforehand. In contrast to Liu et al. we did not enhance the underlying ontology [14]. We therefore experience problems when the necessary ontology knowledge does not exist. Further research has to be put into this problem in the future (see VI).

Another question that came up when working with the ACT thematic relations is that we were not able to distinguish between phrases like "I am eating fish" and "I eat fish". The former would be a UML method as it describes an action. "I eat fish" instead explains determination and should therefore be created as UML relation or -attribute. MIT's ConceptNet [11] is said to deliver a solution to that problem by working with parts of phrases and not only single words.

Also some of the words processed from RCyc needed to be changed to `CycL`-related collections. These were for example the word "called" which was transformed to `ThingsDescribableAsFn` or "commences" which is changed to `BeginningAnActivity` in RCyc. This could not always be automated since the number of options is huge.

V. Related Work

According to Marvin Minsky [15], [4] common sense is the most powerful tool to overcome the problem of losing information when not being able to grasp the semantics and meaning of concepts we humans deal with daily. As mentioned in [11], Marvin Minsky thinks that common sense "is knowing maybe 30 or 60 million things about the world and having them represented so that when something happens, you can make analogies with others".

There is an extensive number of parsers and NLP programs that have acceptable detection rates, but fail when semantics are indispensable for understanding, for example:

- Fred saw the aeroplane flying over Karlsruhe.
- Fred saw the mountains flying over Karlsruhe.

Both times the structure of the sentences is identical. Still the action "flying", that is thematic relation ACT, should be assigned to "the aeroplane" in the former and to "Fred" in the latter.

In previous work we generated GrGen.NET-graphs made of the corresponding thematic relations of the natural language text [3]. We were able to match thematic relations and their meaning to specific concepts in UML models. We would like to support this effort by empowering the machine to use "common sense".

Bethard et al. use timely dependencies when running the classification of possible UML concepts found in natural language text [16]. Liu suggests using extraction of events and their sub-event structure to understand natural language and the connection of phrase concepts better. [14]. Liebermann and Liu argue that there is a high coherence between the English natural language and programming languages [17]. Therefore a direct mapping might be feasible. Natural language processing (NL-processing) is good enough to support natural language programming (NL-programming) since programming still relies on formal languages [18].

Settimi shows that realizing there is no semantical difference between the sentences "the user shall view" and "the system shall display" cannot be corrected with only a thesaurus since it will not be able to distinguish between opposite viewpoints of the same action [19].

Looking at the natural language understanding (NLU) side, Meystre and Haug show that focusing on small domains using highly specified ontologies leads to satisfying results [20], [21], [22].

Using non-specified ontologies on non-limited natural language texts has not been done so far.

VI. Conclusion

This work has examined the post-processing of semantic information already loaded into a graph model and shown that it improves the automatic model creation process. Even organizations with mature software development processes end up with disconnected artifacts in documentation and implementation [23], [24]. Many mistakes stem from faulty designs which tend to lose information during the design phase. We try to avoid this by starting off with the specification itself. We try to make sure the first step into coding is correct by matching the model created from the specification to the code. One could argue that this approach also fails if the requirements documented in the natural language textual specification are incomplete or faulty [1], [2].

Pre-processing specifications before they are represented in a graph and automatic text annotation is another big challenge. We are working on detecting non-functional aspects. It seems that this part of the specification is not necessary for the UML model creation. We also research the detection of "Why?"-clauses/sentences. These are merely to explain why something is done and not how. They are not beneficial for automatic model creation and serve for human motivation and understanding only. Therefore, they can be erased in the pre-processing.

We consider using RCyc's Microtheories and therefore narrowing the search-space within the ontology. This improves detection rates as well as processing speed. Implying RCyc's backward chaining might improve results even more. So far using ontologies to improve model creation has only been realized with the thematic relations AG and ACT. We will extend this functionality to other thematic relations.

As shown in the chess example, there is a need to consider "common sense" when automatically creating models from textual specifications. Though our approach just marks the beginning of using ontologies in combination with natural language processing, it shows that improvements are possible. Considering the fact that it still takes days, weeks or months to completely understand and model a vast textual specification of any system, we assume this approach to be of great help. The area of application is not restricted to software development only as the boundary-setting aspect is the domain knowledge of the ontology.

ACKNOWLEDGMENT

The authors would like to thank CyCorp Inc. for providing ReasearchCyc and supporting the research by answering our questions.

REFERENCES

[1] C. Rupp and R. Goetz, "Psychotherapy for system requirements," *Proceedings of the Second IEEE International Conference on Cognitive Informatics (ICCI '03)*, 2003.

[2] C. Rupp, "Requirements and psychology," *IEEE, May/June 2002*, 2002.

[3] T. Gelhausen and W. F. Tichy, "Thematic Role Based Generation of UML Models from Real World Requirements," in *Proc. International Conference on Semantic Computing ICSC 2007*, 2007, pp. 282–289.

[4] M. L. Minsky, *The emotion machine*. Simon and Schuster, 2006.

[5] O. Denninger, T. Gelhausen, and R. Geiß, "Applications and Rewriting of Omnigraphs – Exemplified in the Domain of MDD," in *Proc. 3rd Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE '07)*, ser. LNCS, A. Schürr, M. Nagl, and A. Zündorf, Eds., vol. NN. Springer, 2008.

[6] T. Parr, "Antlr." [Online]. Available: http://www.antlr.org/

[7] R. Geiß, G. V. Batz, D. Grund, S. Hack, and A. M. Szalkowski, "GrGen: A Fast SPO-Based Graph Rewriting Tool," in *Graph Transformations - ICGT 2006*, ser. Lecture Notes in Computer Science, A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, Eds. Springer, 2006, pp. 383 – 397, natal, Brasil.

[8] T. Gelhausen, B. Derre, and R. Geiß, "Customizing grgen.net for model transformation," in *GraMoT 2008*, 2008.

[9] CyCorp, "Researchcyc." [Online]. Available: http://research.cyc.com/

[10] G. A. Miller, "Wordnet." [Online]. Available: wordnet.princeton.edu

[11] H. Liu and P. Singh, "Conceptnet - a practical commonsense reasoning tool-kit," *BT Technology Journal*, vol. Vol 22, 2004. [Online]. Available: http://larifari.org/writing/BTTJ2004-ConceptNet.pdf

[12] D. Lenat, *Cyc 101 Tutorial*, Cyc.com. [Online]. Available: www.cyc.com

[13] FIDE, *Laws of chess*. FIDE - World Chess Federation, July 2005, no. E.I.01A. Laws of Chess. [Online]. Available: http://www.fide.com/official/handbook.asp?level=EE101

[14] H. Liu, H. Lieberman, P. Singh, and B. Barry, "Beating common sense into interactive applications," 2004. [Online]. Available: http://larifari.org/writing/AIMag2004-BeatingCommonSense.pdf

[15] M. L. Minsky, *The society of mind*. Simon and Schuster, 1986.

[16] S. Bethard, S. Bethard, J. H. Martin, and S. Klingenstein, "Timelines from text: Identification of syntactic temporal relations," in *Proc. International Conference on Semantic Computing ICSC 2007*, J. H. Martin, Ed., 2007, pp. 11–18.

[17] H. Liu and H. Liebermann, "Toward a programmatic semantics of natural language," p. 2, 2004.

[18] H. Liu, H. Lieberman, and R. Mihalcea, "Nlp (natural language processing) for nlp (natural language programming)," *A. Gelbukh (Ed.) (c)Springer-Verlag Berlin Heidelberg 2006*, vol. CICLing 2006, LNCS 3878, pp. pp. 319–330, 2006. [Online]. Available: http://larifari.org/writing/CICLING2006-NLP4NLP.pdf

[19] R. Settimi, O. Cleland-Huang, J.and Ben Khadra, J. Mody, W. Lukasik, and C. DePalma, "Supporting software evolution through dynamically retrieving traces to uml artifacts," *Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of*, pp. 49–54, 2004.

[20] S. Meystre and P. J. Haug, "Medical problem and document model for natural language understanding," *AMIA Annu Symp Proc*, pp. 455–459, 2003. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1479970

[21] ——, "Automation of a problem list using natural language processing." *BMC Med Inform Decis Mak*, vol. 5, no. 30, August 2005. [Online]. Available: http://dx.doi.org/10.1186/1472-6947-5-30

[22] Meystre and Haug, "Natural language processing to extract medical problems from electronic clinical documents: Performance evaluation," *J. of Biomedical Informatics*, vol. 39, no. 6, pp. 589–599, 2006.

[23] P. Arkley, P. Mason, and S. Riddle, "Position paper: Enabling traceability," *Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering*, vol. Edinburgh, Scotland (September 2002), pp. 61–65, 2002.

[24] G. Antoniol, G. Canfora, G. Casazza, and A. D. Lucia, "Information retrieval models for recovering traceability links between code and documentation," in *In Proceedings of IEEE International Conference on Software Maintenance, San Jose, CA, 2000*, 2000.

Ontology-based Development of Testing Related Tools

Ellen F. Barbosa, Elisa Y. Nakagawa, Ana C. Riekstin, José C. Maldonado
University of São Paulo – ICMC/USP
Av. do Trabalhador São-Carlense, 400
P.O. Box 668, 13560-970 – São Carlos (SP), Brazil
{*francine, elisa, claudiar, jcmaldon*}*@icmc.usp.br*

Abstract

Testing constitutes one of the most relevant software engineering activities in order to guarantee the quality of the software under development. Besides that, the automation of testing activity is an important issue to be addressed – the availability of testing tools makes the testing a more systematic activity and minimizes the errors caused by human intervention. In spite of their advantages, most of testing tools are still built in an ad hoc way and considerable rework is necessary if it is desired to configure or adapt them to different testing techniques and criteria. In this paper we discuss the using of an ontology of software testing (OntoTest) in the development of testing related tools. Two perspectives have been investigated: (i) the development of an OntoTest-based application to share and harmonize testing concepts; and (ii) the specification of a common set of functional modules that testing tools should provide, aiming at establishing an OntoTest-based testing architecture.

Keywords: *Ontology, Software Testing, Testing Tool.*

1. Introduction

Producing reliable, robust and high quality software systems is one of the most important software development concerns. Testing, which intends to reveal the existence of faults in the software, constitutes one of the most relevant software engineering activities, being crucial to guarantee the quality of the software under development. Besides that, the data collected during testing phases is also important for debugging, maintenance, and reliability assessment [11].

The software testing domain involves integration of three basic types of knowledge – theoretical, empirical and tool specific. In fact, a significant amount of information should be mastered to perform an effective testing activity (e.g., testing techniques and criteria, testing phases, testing steps, testing artifacts, testing tools). Such diversity of concepts and their inter-relations make the establishment of a consensual shared understanding on software testing a fundamental issue to be addressed.

Ontologies play an important role in this perspective. An ontology is a formal and declarative representation which includes [6, 13]: (i) the vocabulary required for referring to the concepts in the subject area; and (ii) the logical statements which describe what the concepts are and how they are related to each other. Hence, it provides a vocabulary for representing and communicating knowledge about some topic as well as a set of relationships which hold among the concepts in that vocabulary.

Currently we are establishing *OntoTest* [2], an ontology of software testing, which aims to support acquisition, organization, reuse and sharing of knowledge on the testing domain. Based on ISO/IEC 12207 [9] and on the Falbo and Bertollo's work [4], *OntoTest* intends to explore different perspectives involved in the testing activity, such as techniques and criteria, human and organizational resources, and automated tools.

Also regarding the quality and productivity of the software development process, the automation of testing activity is another important issue to be addressed. Indeed, applying a testing criterion without the support of a testing tool is an error-prone activity.

In order to automate software testing, commercial and academic testing tools have been developed in the last years. Despite their advantages, most of testing tools are still built in an ad hoc way. As a consequence, is difficult to identify a well-defined architecture for them. Besides, since a testing tool typically supports a unique testing criterion, considerable rework is necessary if it is desired to configure or adapt it to another criterion or technique.

In this paper we discuss the using of *OntoTest* in the development of testing related tools. Two scenarios are considered: (i) to developing an *OntoTest*-based application for organizing and searching testing concepts, aiming at disseminating testing information; and (ii) to defining a common set of functional modules that different testing tools

The authors would like to thank the Brazilian funding agencies (FAPESP, CAPES, CNPq) and to the QualiPSo Project for their support.

should provide, aiming at establishing an *OntoTest*-based architecture for testing tools.

The remainder of this paper is organized as follows. In Section 2 the related work is presented. Section 3 describes *OntoTest* and its general structure. In Section 4 we discuss how testing related tools can be built based on *OntoTest*. In Section 5 we summarize our contributions and discuss the perspectives for further work.

2. Related Work

An ontology is a formal explicit specification of a shared conceptualization [6]. That is, a simplified way of perceiving a piece of reality, often conceived as a set of relevant terms and their relationships, whose structure is constrained by some rules. Basically, it consists of concepts and relations, as well as their definitions, properties and constraints expressed by means of axioms [13].

Ontologies have been applied to describe a variety of domains, such as medicine, engineering and law. In software engineering, several ontologies have already been identified – software engineering ontology [15, 16], software quality ontology [1], software process ontology [4], enterprise ontology [14], software testing ontology [7, 17], software maintenance ontology [10], among others.

Falbo and Bertollo [4], for instance, developed an ontology of software process to support the acquisition, organization, reuse and sharing of software process knowledge. The ontology is designed to support software process definition and automatization in a meta-SEE (Software Engineering Environment), which is able to generate, by means of instantiation, SEEs adequate to the particularities of specific software processes, application domains and projects. Every knowledge based tool linked to the software process ontology shares a common vocabulary, facilitating the communication between developers and, also, allowing the sharing and reuse of knowledge bases in the meta-environment as much as in the instantiated SEEs.

Regarding software testing, Huo et al. [7, 17] investigated the development of an ontology of testing as a support for a multi-agent software environment for testing web-based applications. The idea is to use the ontology to enable flexible integration and to mediate communication between multiple agents. In short, a taxonomy of testing concepts (basic and compound concepts) is established. The ontology is represented in UML, at a high level of abstraction, and in XML to codify the knowledge of software testing for agents' processing of messages.

3. OntoTest: an Ontology of Software Testing

Most of the testing concepts considered in Huo's work [7, 17] are in agreement with the concepts established in *OntoTest* [2]. The software process ontology, defined by Falbo and Bertollo [4], and the ISO/IEC 12207 standard [9] were also considered, focusing on the idea of a testing process. Additionally, we have explored concepts from: (i) definition and evaluation of testing criteria; and (ii) knowledge and experience in testing tools development.

Due to the complexity of the testing domain, we have adopted a layered approach to the development of *OntoTest*. On the ontology level, the Main Software Testing Ontology addresses the main concepts and relations associated with testing. On the sub-ontology level, specific concepts from the Main Software Testing Ontology – testing process, testing artifacts, testing steps, testing strategies and procedures, and testing resources – are refined and treated into details. Figure 1 shows the graphical representation of the main ontology. We adopted UML notation for representing it.

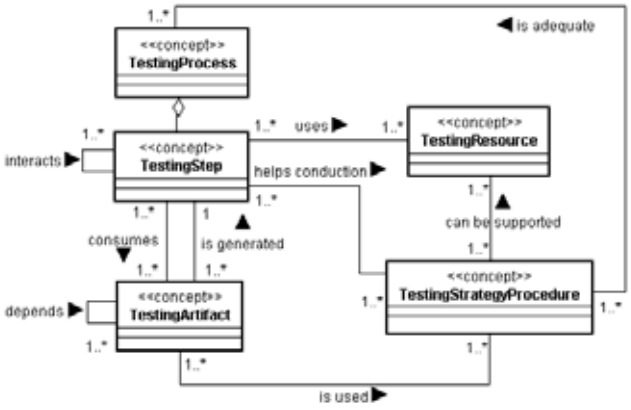


Figure 1. Main Software Testing Ontology

To develop the Main Software Testing Ontology, we established an analogy between software process and software testing process [9, 4]. Similar to the software process, which is a sequence of steps required to develop and maintain software, a testing process can be seen as a sequence of testing steps required to develop and maintain the testing activity. A testing step can consume and/or produce several testing artifacts* and can use different testing resources. Moreover, when defining a testing process, it is important to determine how the testing steps shall be performed. To do so, we must establish the testing strategies and procedures, adopted in the accomplishment of the testing steps.

For each basic concept represented in the Main Software Testing Ontology, there may be a number of subconcepts which are refined and treated in the sub-ontologies [2]:
Testing Process: Testing processes are defined based on the development paradigm as well as on the application do-

*A testing step can also consume some artifacts produced by other activities of the software development process (e.g., a requirement specification document, a quality plan, and so on).

main. Testing life cycle models are also used as a reference in the definition of a testing process, establishing its main testing steps and the dependency relations among them.

Testing Step: When performing a test, a set of essential steps should be considered [11]. Each testing step is composed of testing activities. Based on the ISO/IEC 12207 standard [9], we have classified a testing activity as primary, organizational or supporting one, depending on the role it plays in the testing process. As primary activities of software testing we have identified: (1) test case design; (2) artifact under testing handling; (3) test requirement establishment; (4) test execution; and (5) test analysis and measurement. Similarly, organizational and supporting testing activities have also been established.

Testing Artifact: Each testing step may consume and/or produce different testing artifacts (test documents, test cases and test requirements). In terms of test documentation, eight different documents, established on the basis of the IEEE 829 standard [8], are represented in the sub-ontology. Test cases consist in the input data against which the software is executed, in conjunction with the output data observed [11]. Test requirements correspond to the required elements to be exercised during the testing activity [11].

Testing Strategy Procedure: Different testing strategies and procedures can be established according to the artifact under testing (system requirement, specification, design, source code), and depending on how testing phases (unit, integration, system or regression testing), testing approaches (requirement-based, specification-based, design-based or implementation-based) and testing techniques (functional, structural, error-based, state-based, random or ad-hoc) are combined. Testing techniques are responsible for establishing the testing criteria to be adopted. Control-Flow (e.g., All-Nodes, All-Edges, All-Paths) [11], Data-Flow (e.g., All-Definitions, All-Uses) [12], and Mutation Analysis [3] are some of the representative criteria that can be found in the literature. Also, testing techniques support testing methods, which are based on testing guidances (standards and guidelines).

Testing Resource: Testing resources are required to the accomplishment of a testing step. The Testing Resource Sub-Ontology is presented in details in Section 4.

OntoTest has been described in OWL † (Web Ontology Language), using the Protégé ‡ ontology development tool.

4. OntoTest and Testing Related Tools

We have explored *OntoTest* in two scenarios: (i) to developing a tool to disseminate and harmonize testing concepts; and (ii) to structuring an ontology-based architecture for testing tools.

†http://www.w3.org/TR/owl-ref/

‡http://protege.stanford.edu

4.1. An OntoTest-Based Application

Figure 2 illustrates *OntoTestSearchingTool* – a web application, based on *OntoTest*. It allows searching for testing concepts, showing concept's details such as: (i) description, (ii) direct superclasses and subclasses (concept hierarchy); (iii) *has part* and *is part of* relations (concept composition); (iv) images; (v) facts (additional information related to the concept); (vi) references; and (vii) examples. Figure 2 shows the results of a query on the concept *TestingTool*.

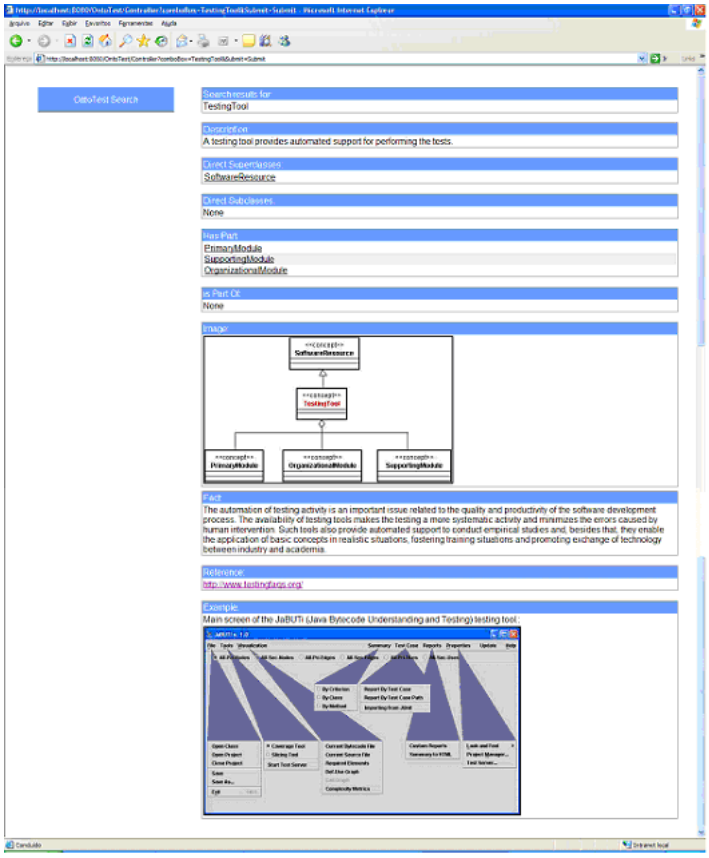


Figure 2. OntoTestSearchingTool

As a short-term goal, we intend to explore *OntoTestSearchingTool* for teaching/learning software testing. In this sense, the *OntoTest*-based application would be explored as part of a testing tool, in particular as an organizational module for training activities.

4.2. An OntoTest-Based Set of Testing Modules

According to the sub-ontologies of *OntoTest*, a testing step can be seen a transformational primitive where step inputs and outputs correspond to testing artifacts. Nevertheless, other elements are necessary to the accomplishment of a testing step. These elements are called testing resources

Figure 3. Testing Resource Sub-Ontology

and are considered in the Testing Resource Sub-Ontology (Figure 3).

A testing resource can be a human, a hardware or a software resource. Hardware and software resources are characterized in terms of a testing environment, which can be used to automate the testing strategies and procedures. Testing tools, which provide automated support for performing the tests, are a special kind of software resources. Considering the ISO/IEC 12207 standard, three types of functional modules can be considered as part of a testing tool: (i) primary, (ii) organizational, and (iii) supporting modules.

Based on our experience in testing tools development and on the primary activities of software testing established in the Testing Step Sub-Ontology, we have identified and represented in *OntoTest* a set of primary functional modules that a testing tool should provide:

Test Case Module: Comprises the basic operations involving the test case set (create, view, update, store, remove, import/export, enable/disable and minimize). For instance, test cases can be manually or automatically created, or can be imported from text documents or other testing tools. They can be enabled/disabled in a test session without to be physically removed from the test case database. Also, the test case set can be minimized in order to keep only the most effective test cases.

Artifact Under Test Module: Comprises the basic operations involving the artifact under testing (acquire, analyze, instrument, and store). Depending on the testing approach, an artifact under testing can be a source code, a system requirement, a specification or a design document. In particular, instrumentation is a technique frequently used in software testing for different purposes, (e.g., program and/or specification execution trace, and testing criteria coverage analysis). Instrumenting the artifact to be tested can be divided into two main tasks: (i) deriving the artifact structure; and (ii) inserting statements for collecting runtime/simulation information.

Test Requirement Module: Comprises the basic operations involving the test requirements (establish/generate, execute, enable/disable, mark/unmark requirements as infeasible, and store). Test requirements are the required elements to be exercised during the tests to satisfy a given testing criterion. A requirement is said infeasible if it cannot be exercised by any input data and should be discarded.

Test Execution Module: Responsible for executing the test cases against the artifact under test.

Test Analysis and Measurement Module: Responsible for determining the percentage of satisfaction of the test requirements for a specific criterion by a test case set. It also generates statistical reports (e.g., number of test requirements exercised/not exercised, the most effective test cases, and so on) about the performed tests.

Supporting and organizational modules have been specified similarly. In short, the automated support they provide is in agreement with the processes established by ISO/IEC 12207, adapted to the scope of software testing. As supporting modules, a testing tool should provide mechanisms for automating test documentation and test configuration management, among others. In terms of organizational modules, it should include mechanisms for automating the test

training activities, for instance. It is important to highlight that ontology-based tools, such as *OntoTestSearchingTool*, can also be integrated to testing tools as organizational modules for training.

Aiming at a basis to develop testing tools as well as to systematize its construction in a testing architecture, we have refined the Testing Resource Sub-Ontology, discussing how testing tools to support data-flow [12] and mutation testing [3] can be built based on this sub-ontology. Moreover, this refinement helped us: (i) to validate the concepts and relationships of the Testing Resource Sub-Ontology; and (ii) to investigate how difficult is the establishment of more specific testing ontologies from *OntoTest*.

Basically, we noticed that *Test Case* and *Source Code Under Test* modules remain the same for all testing subdomains. The *Test Requirement Module* corresponds to the *Association Module* for data-flow, and to the *Mutant Module* for mutation:

Association Module: The data-flow test requirements correspond to the associations, i.e., interactions between each variable definition and its subsequent uses (or references) through the program. To derive the associations it is necessary to construct the def-use graph, which represents the definitions and uses of all variables of the program.

Mutant Module: The mutation test requirements corresponds to the mutants, i.e., different versions of the original program, each of which containing a simple syntactic change (fault). The simple faults are modeled by a set of mutant operators applied to a program under testing. A mutant can also be enabled/disabled through a given test session. So, this module is responsible for generating and handling the set of mutants.

Regarding the *Test Execution Module*, it corresponds to the *Instrumented Source Code Execution Module* for data-flow, and to the *Test Execution Module* for mutation:

Instrumented Source Code Execution Module: Executes the test cases against the instrumented program generated by the *Source Code Under Test Module*.

Test Execution Module: For mutation testing, it is necessary to execute the test cases not only against the original program but also against the set of generated mutants. The quality of the test set is measured by its ability to distinguish the behavior of the mutants from the behavior of the original program. The goal is to find a test case that causes a mutant to generate a different output from that of the original program. This kind of mutant is said to be dead. In this sense, the *Test Execution Module* is composed by two other sub-modules: (1) the *Mutant Execution*, responsible for executing the mutants; and (2) the *Source Code Under Test Execution*, responsible for executing the original program.

Finally, the *Test Analysis and Measurement Module* corresponds to the *Data-Flow Analysis and Measurement Module* for data-flow, and to the *Mutation Analysis and Measurement Module* for mutation:

Data-Flow Analysis and Measurement Module: Responsible for providing mechanisms for: (1) determining the infeasible associations (associations which cannot be exercised by any test case); (2) coverage analysis (determining the percentage of satisfaction of the associations for a given data-flow criterion by the test case set); and (3) generating statistical reports about the performed tests.

Mutation Analysis and Measurement Module: Responsible for providing mechanisms for: (1) determining the equivalent mutants – mutants that always generate the same output from that of the original program; (2) calculating the mutation score – the ratio of the number of dead mutants to the number of non-equivalent mutants; and (3) generating statistical reports about the performed tests.

It is also important to notice that the training organizational modules to be considered in data-flow and mutation testing tools can be obtained by instantiating *OntoTest* for each specific knowledge domain. *OntoTest*-based applications, similar to *OntoTestSearchingTool*, can be generated and integrated as training modules into the testing tools.

For the sake of illustration, Figure 4 shows part of the Mutation Testing Resource Sub-Ontology.

Although supporting different testing techniques and criteria, both data-flow and mutation testing tools presented a very similar set of functionalities. This result reassures our expectation on using the Testing Resource Sub-Ontology as a supporting mechanism to define the basic functionalities a testing tool should provide. Actually, from the functional modules defined in the sub-ontology, specifying the core for a testing tool seems to be straightforward. The common set of functional modules for testing tools we have identified should be further investigated as a basis for the establishment of an ontology-based testing architecture.

5. Conclusions and Further Work

In this paper we discussed the using of an ontology of software testing in the development of testing related tools. Two perspectives were explored: (i) the development of an *OntoTest*-based application for sharing and harmonizing testing concepts; and (ii) the refinement of *OntoTest* aiming at investigating how testing tools can be built based on this ontology.

Currently, we are investigating the establishment of an ontology-based architecture to support the development of tools to automate the testing activities. The testing architecture is being built upon the *OntoTest* concepts and relationships and can be refined to different testing domains, supporting the development of specific testing tools. Also, we observed that current testing tools do not present considerable degree of integration, including data integration.

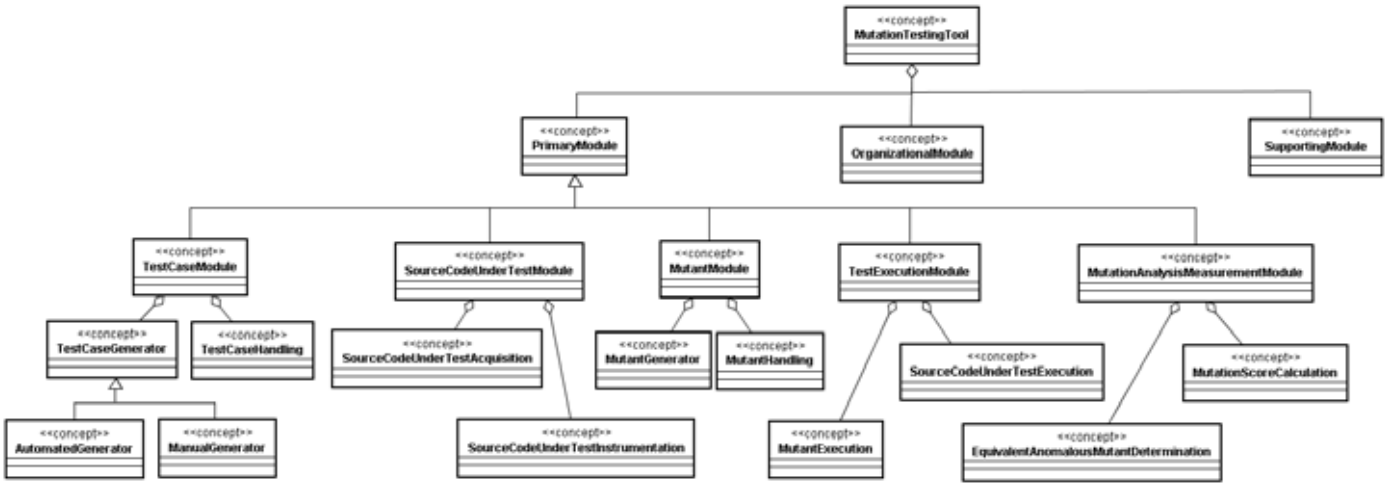


Figure 4. Part of the Mutation Testing Resource Sub-Ontology

On the other hand, intelligent agents have recently been investigated as a mechanism to test software. *OntoTest* should be further explored to enable integration of testing tools and of multiple agents by sharing knowledge, transferring information and negotiating the agent's actions.

Finally, since the body of knowledge on testing keeps evolving, mainly due to the experimental studies conducted, we are also developing an ontology for experiments on software engineering ($_{EXPER}Ontology$ [5]), which should be integrated to *OntoTest*. Both ontologies should be explored in the context of testing reference architectures. At the very end, the idea is to compose, in an incremental and evolutionary way, a software testing environment, integrating tools, processes, artifacts and experimental studies, being useful in promoting reuse of testing expertise and in achieving well-recognized understanding in testing areas. Our experience on using ontologies in the establishment of an architecture-based software testing environment should be presented in a forthcoming paper.

References

[1] T. H. Al Balushi, P. R. F. Sampaio, D. Dabhi, and P. Loucopoulos. Performing requirements elicitation activities supported by quality ontologies. In *18th Int. Conference on Software Engineering and Knowledge Engineering (SEKE 2006)*, pages 343–348, San Francisco, CA, July 2006.

[2] E.F. Barbosa, E. Y. Nakagawa, and J.C. Maldonado. Towards the establishment of an ontology of software testing. In *18th Int. Conference on Software Engineering and Knowledge Engineering (SEKE 2006)*, pages 522–525, San Francisco, CA, July 2006. Short Paper.

[3] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):34–43, April 1978.

[4] R. A. Falbo and G. Bertollo. Establishing a common vocabulary for software organizations understand software processes. In *EDOC Int. Workshop on Vocabularies, Ontologies and Rules for The Enterprise (VORTE 2005)*, Enschede, The Netherlands, September 2005.

[5] R. E. Garcia, E. N. Höhn, E.F. Barbosa, and J.C. Maldonado. An ontology for experiments on software engineering. In *20th Int. Conference on Software Engineering and Knowledge Engineering (SEKE 2008)*, San Francisco, CA, July 2008. To appear.

[6] T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. *Int. Journal Human-Computer Studies*, 43(5/6), 1995.

[7] Q. Huo, H. Zhu, and S. Greenwood. A multi-agent software environment for testing web-based applications. In *27th Annual Int. Computer Software and Applications Conference (COMPSAC03)*, 2003.

[8] IEEE Software Engineering Technical Committee. Standard for Software Test Documentation, September 1998.

[9] International Organization for Standardization. ISO/IEC 12207. Information technology – software life-cycle processes, 1995.

[10] B. Kitchenham, G. H. Travassos, and A. Maryhauser. Towards an ontology of software maintenance. *Journal of Sofware Maintenance: Research and Practice*, 11(6):365–389, 1999.

[11] G. J. Myers, Corey Sandler, Tom Badgett, and Todd M. Thomas. *The Art of Software Testing*. John Wiley & Sons, 2nd. edition, 2004.

[12] S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, SE-11(4):367–375, April 1985.

[13] M. Uschold and M. Grüninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2), June 1996.

[14] K. Villela, G. Santos, L. Schnaider, A. R. Rocha, and G. H. Travassos. The use of an enterprise ontology to support knowledge management in software development environments. *Journal of Brazilian Computer Society*, 11(2):45–59, November 2005. Special Issue on Ontologies Issues and Applications.

[15] P. Wongthongtham, E. Chang, T. S. Dillon, and I. Sommerville. Software engineering ontology – the instance knowledge (part i). *International Journal of Computer Science and Network Security*, 7(2):16–26, February 2007.

[16] P. Wongthongtham, E. Chang, T. S. Dillon, and I. Sommerville. Software engineering ontology – the instance knowledge (part ii). *International Journal of Computer Science and Network Security*, 7(2):27–36, February 2007.

[17] H. Zhu and Q. Huo. Developing a software testing ontology in UML for a software growth environment of web-based applications. In *Software Evolution with UML and XML*. Idea Group, 2004. Hongji Yang (eds.).

Test Order Generation for Efficient Object-Oriented Class Integration Testing

Rattikorn Hewett & Phongphun Kijsanayothin
Dept. of Computer Science, Texas Tech University
rattikorn.hewett@ttu.edu, kphongph@gmail.com

Darunee Smavatkul
Dept. of Computer Science, Chiangmai University
darunee@chiangmai.ac.th

Abstract

Testing object-oriented software is complex and costly. A strategy to incrementally testing and integrating components (classes or methods) in object-oriented software while minimizing the number of test stubs has been proposed. This reduces testing efforts and thereby reduces testing cost and enhances testing efficiency. However, an important issue of how to select an appropriate component test order remains. This paper presents a new approach to test order generation that requires (near) optimal number of test stubs by exploiting heuristics based on a dependency graph constructed from a given object-oriented model. Our approach offers several benefits. First, it is simpler and more efficient than most other graph-based approaches with an improved complexity of quadratic polynomial time in the number of components. Second, it is deterministic and thus, the resulting test order is not biased by randomness. Finally, it is flexible in that it can be applied to object-oriented models at different levels of detail. The paper describes the proposed approach and gives two illustrations, one of which is a case study of an application system in telecommunication.

1. Introduction

Software testing is primarily concerned with software behaviors. Testing object-oriented software is a challenging task. Unlike conventional software that is functionally decomposed into separate procedures, typical object-oriented software does not offer a clearly defined behavioral composition structure (despite the enforcement of polymorphisms for component reuse). Furthermore, there is no implication of execution order from sequence of object-oriented codes. A large number of possible interactions, from message passing between class instances or objects depend on their dynamically changing states. This makes it impossible or extremely costly to test all possible interactions.

Much research has developed many techniques and strategies for testing object-oriented software at various levels (e.g., unit, integration or cluster, and system levels) [1, 4, 6, 8]. Incremental strategies to testing and integrating components (class or methods) in object-oriented software have been proposed to minimize the number of stubs [2, 7, 10, 11] as well as to execute complete end-user functionalities [1, 6, 8]. This paper concentrates on the former. When testing a component that depends on other components that have not yet been developed or tested, a tester has to develop stubs to emulate these other components that the component under testing depends on. Stubs are often recognized as a major cost of software testing, therefore minimizing the needs for stubs can reduce testing cost and enhance testing efficiency. However, an important issue of how to select an appropriate component test order remains.

Several graph-based algorithms for deriving a class integration test order to minimize stubbing from dependencies in various forms of object-oriented models have been proposed [2, 4, 7, 10, 11]. Kung et al. [7] and Traon et al. [11] use a test dependency graph constructed from a UML (The Unified Modeling Language) class diagram [9], whereas Tai and Daniels [10] and Briand et al. [2] employ an object relational diagram with dependency of three types: aggregation, inheritance and associations [9].

All of the above techniques share the same basic idea to obtain a class integration test order by applying a topological sorting to a given dependency graph. Unfortunately, the application of the depth first search (DFS, specifically the Tarjan algorithm) to topological sorting is only applicable to an acyclic directed graph [3]. Thus, these techniques use a top-down approach to first decompose a given dependency graph into a tree-like decomposition, where each node is a cluster of components that may contain cycles. Next is to remove appropriate dependencies to "break" the cycles in these clusters. Repeat these steps until there is no cycle in the clusters so that a topological sorting can be applied in order to obtain a partial solution of a class test order. By tracing along the tree-like decomposition in an upward manner from each partial list, a complete class integration test order can be obtained. In [2, 7, 11], the clusters are identified with strongly connected components (SCCs) in the dependency graph. In [10], the clusters are classes in the same hierarchical levels of inheritance and aggregation relationships (referred to as *major levels* in [10]). Because cycles are likely to occur in a class diagram when software development evolves to later stages, resolving the cycles (so that a topological sorting can be applied) becomes a central issue.

This paper presents a new approach to select the order of classes in object-oriented software to be tested and integrated incrementally in such a way that the required number of stubs is (near) minimal. Our approach is fundamentally different from previous graph-based approach in that instead of decomposing the dependency graph into clusters and concentrating on removing dependencies to "break" cycles in each cluster, we use a bottom up approach to select good candidates to be tested and filter out undesirable

ones by employing appropriate heuristics based on the dependency graph and the objective of the strategy. The algorithm gains efficiency by directly generating a class test order based on the merit of each component toward the goal to minimize the number of stubs instead of worrying about its abstraction or decomposition level.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 discusses how to construct a test dependency graph from an object-oriented model, specifically a UML class diagram. Section 4 describes the proposed algorithm, our main contribution. Section 5 illustrates our approach with some comparisons with previous approach using two examples, one of which is a case study of a real-world application in telecommunication. The paper concludes in Section 6.

2. Related Work

There are various incremental strategies for testing and integrating object-oriented software components with different objectives. Some focus on behavioral testing that aims to execute complete end-user functionalities by tracing method/message paths using state-based, event-based approaches or use cases [1, 6, 8]. Some focus on class interaction testing that aims to minimize the number of stubs [2, 7, 10, 11]. This paper addresses the class test order problem of the latter type.

A variety of existing graph-based solutions to the class test order problem can be characterized by different techniques proposed to "break" the dependency cycles. Kung et al.[7] introduced an initial basic solution by applying a topological sorting to a given (acyclic) dependency graph and simply selecting a random dependency to break cycles in each cluster (or SCC). Tai and Daniels' approach [10] makes use of two dependency types (aggregation and inheritance) to group components into clusters (or major levels). It then breaks cycles in each cluster by removing an association dependency (say, from class *A* to class *B*) with the highest sum of *n,* the number of incoming dependencies of *A* and *m,* the number of outgoing dependencies of *B*. The rationale is to break an association that will likely break a larger number of cycles. The approach presented by Traon et al. [11] uses the DFS algorithm to identify clusters in terms of SCCs. As a result, it identifies *back* edges [3] in each SCC. To break cycles, remove all the incoming back edges to the (root) component that has the highest sum of incoming and outgoing back edges. The algorithm is called recursively for each nontrivial SCC. Briand et al.'s strategy [2] combines the two techniques of [10, 11] by recursively identifying SCCs (like [11]) and removing an association dependency with the highest *product* (not the *sum* in [10]) of *n* and *m* as defined above.

The study by Briand et al.[2] shows that Tai and Daniel's technique can result in unnecessary stubbing, while Traon et al.'s approach may lead to breaking cycles by removing aggregation or inheritance dependencies which would require stubbing of all parent classes which are not economically viable. Traon et al.'s technique appears to give optimal solution for non-specific stubs, while Briand et al.'s approach performs well on specific stubs (see definitions in Section 4.3). The computation for identifying SCCs recursively takes $O(n^3)$ time (see Section 4.2), where *n* is the number of components in the dependency graph. Unlike these techniques, our approach requires neither the identification of SCCs (i.e., clusters) nor topological sorting. Therefore, it reduces computational cost and eliminates non-determinisms that could occur in these steps.

3. From UML to Test Dependency Graph

This section describes how to map from a given object-oriented (OO) structural model (class diagram) into a corresponding test dependency graph. We use UML [9], a common standardization of semi-formal OO modeling language for representing the structural models of OO software. Our technique is based on that of Traon et al. [11] but our mappings are more direct and more generic to provide easy extensibility to other types of OO models. Since the test dependency graph construction technique is not our main focus, we briefly describe the technique and basic mechanisms for completeness. For more details, see [11].

Let $G = (V, E)$ be a test dependency graph, where each node in V represents a software *component* (class or method) and each directed edge (u, v) in E represents a *test dependency* of component u on component v (u depends on v for testing). E can be constructed by syntactically mapping each *relationship* (including a *dependency* that can be inferred) among components in the UML model to a corres-

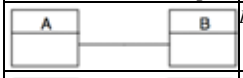
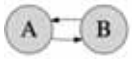
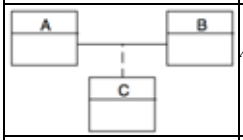
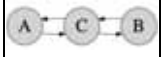
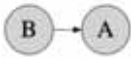
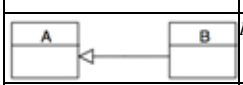
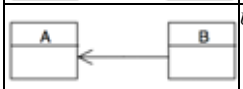
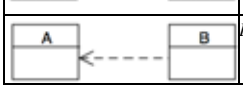
UML Relationship	Meaning	Dependency
	*Bi-direction:* A is associated with B and B is associated with A	
	Aggregation: B is a part of A and is not destroyed if A is	
	Composition: B is a part of A and is destroyed if A is	
	*Association Class:* C is an association between A and B	
	Qualified association: B uses a qualifier (e.g., A's index) from A to identify its relationship with A	
	*Inheritance:* B inherits A's properties (A generalizes B)	
	Realization (*implementation*)*:* B relies on A to implement the specification it provides	
	*Uni-direction* (*navigability*)*:* B is associated with A (or B needs A's reference to navigate to A) - permanent	
	*Dependency:* B uses (or has knowledge) of A - non-permanent	

Fig 1 Mappings in class level.

ponding test dependency between nodes in *G*. Different types of mappings are categorized in three levels of application, namely *high-level design, low-level design,* and *code level* depending on the levels of detail in the UML model.

In the high-level design, each class in the UML class diagram does not provide any information about the class *method.* The class mainly contains class name, attributes and relationships to other classes. Thus, the high-level design model contains *class-to-class relationships* that can be mapped directly to *class-to-class test dependencies* among class components in *G*. Figure 1 shows a variety of mappings in this level. Here the top three UML relationships: *bi-directional associations, aggregations* and *compositions* can be mapped to a two-way test dependency between classes. The *association class* relationship corresponds to a two-way test dependency with an additional class in between. The rest of the UML are directional, namely *qualified association, generalization, realization* (*implementation*), *dependency,* and *uni-directional* (*navigational*) *association.* Thus, they can be mapped into one-way test dependencies between classes.

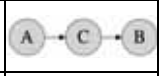
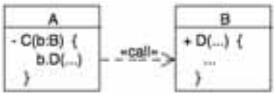
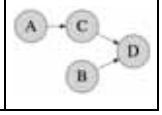
UML Dependency	Meaning	Dep. Graph
	Parameter Dependency: method C of A has a parameter of type B	
	*Call Dependency:* method C of A calls method D of B. C is private (to A) but D is public.	

Fig 2 Mappings in more refined levels.

In the next two levels of mapping applications, the UML model is specified in more details. Classes include more information about methods and specific different types of dependency relationships (e.g., *parameter, call, send, import, instantiate, bind*) [9] can be specified. These details can be exploited to infer more test dependencies. In the low-level design, a class contains *method signatures* (e.g., method names, parameters and types). Thus, the inferred dependency relationships between classes and methods in the UML model can be mapped to *class-to-method test dependencies.* The top part of Figure 2 shows one example of such mappings. Here class *A* contains a signature of method *C* whose parameter *b* has a data type of class *B*. Thus, the dependency relationship is annotated by «*parameter*» as a specific dependency relationship. We can infer that testing *A* depends on *C* and *C* on *B* giving a test dependency graph on the top right of Figure 2.

It is possible that we may want to integrate a new component into a legacy system. In such a case, the OO model can be viewed at the code level. In this level, a class also includes *method implementations* in addition to signatures. Methods can be declared for different levels of visibility. For a *public* method *m* (denoted by +*m*), any component can call *m.* However, if *m* is a *private* method (denoted by −*m*) then *m* can only be called within its class, whereas a *protected* method can be called by any subclass within its class. The bottom part of Figure 2 gives an example of the mapping in this level. Here class *A* contains a method *C* whose implementation calls a method *D* of class *B* (as specified between the curly brackets after the signature of method *C*). This implies that testing *A* depends on *C* whose testing depends on *D,* a component which *B* depends on as shown as a dependency graph on the right of the figure. Note that the mappings in this level can involve *method-to-method test dependencies.* It is also possible to have a class containing a method that calls itself.

The three types of the above mappings should be applied to the UML model at the lowest level possible. This is to prevent redundant test dependencies that can impact stub development and testing cost. For example, in the example at the bottom of Figure 2, if we applied a low-level design mapping from a signature of method *C,* which has a parameter of class *B*, we would have added a test dependency link from *C* to *B*. Therefore, testing *C* depends on not only *D* but also *B* as well. While this is true, the additional test dependency requires unnecessary efforts. Since class *B* contains a method *D*, building stub to simulate *D* requires less work than developing stub for *B.* The latter is not necessary since *C* depends on only parts of *B* (namely *D*), which is already included for the integration testing.

4. Generating Integration Test Order

4.1 Proposed algorithm

This section describes our algorithm for generating a component (class or method) test order from a given test dependency graph described in Section 3. The goal of our test strategy is to order the components to be tested and integrated incrementally so that a total number of stubs required for overall testing is (near) minimal.

For a given software component *n* in the dependency graph, let *out*(*n*) be a set of target nodes of all outgoing edges from *n* (i.e., components that *n* depends on) and *in*(*n*) be a set of source nodes of all incoming edges to *n* (i.e., components that depend on *n*). The cardinality of *out* and *in* is commonly known as an *out-degree* and *in-degree*, respectively [3]. To generate a component test order that satisfies the strategy goal, our proposed algorithm employs two heuristic functions: *h*(*n*) and *f*(*n*). The former estimates a set of stubs required for testing a component *n* and the latter estimates a current set of components that need *n* for testing. While *h*(*n*) directly impacts the goal, *f*(*n*) indicates a degree of usage of a stub that simulates *n*. Figure 3 shows basic steps in our algorithm.

Let *S* and *T* be a set of stubs and a sequence of components tested so far, respectively. To test component *n,* we need a stub for each node that *n* depends on and that has neither been developed nor tested. Therefore, *h*(*n*) contains nodes that are in *out*(*n*) but not in *S* or *T.* The algorithm first selects a component that requires no stubbing or requires the smallest number of stubs (i.e., $|h(n)| > 0$). If there is more than one such component in the latter case (i.e., those

```
Input:    A test dependency graph G = (V, E)
Outputs:  T, a sequence of class order for integration testing
          S, a set of stubs

T ← < >, S ← ∅
While T does not contain all nodes in V do
   For each n in V do ; find nodes with minimum required stubs
        h(n) ← {m ∈ out(n) | m does not appear in T and m ∉ S}
   if there is a node k such that |h(k)| = 0 or a unique node k such
      that |h(k)| = min_{i∈V} |h(i)| > 0
      then Select(k)
      else ; eliminate dependent nodes
         M ← {n | |h(n)| = min_{i∈V} |h(i)|}
         P ← ∅;
         For each u, v ∈ M
            if u ∈ out(v) but v ∉ out(u) then P ← P ∪ {v}
               ; if v depends on u and not viz., eliminate v
         if P ≠ M then M ← M – P
         For each n in M do ; find nodes with maximum usage
               f(n) ← {m ∈ in(n) | m is not in T}
         k ← the first node found such that |f(n)| = max_{i∈M} |f(i)|
         Select(k)
Return(T, S)

Select(n)
  Append n to T
  S ← S ∪ (out(n) – T) ; add nodes that n depends on but have
                       ; not been tested yet
  V ← V – {n} ; update set of nodes that remain to be tested
```

Fig 3 Algorithm for generating a component test order.

in *M* of Figure 3), if the minimum number of the required stubs is zero, select the first component found in *M* to be a candidate for integration testing. On the other hand, if the minimum number of the required stubs is non-zero then the algorithm uses two additional rules to select, from *M,* an appropriate component to be tested. The first rule uses a dependency constraint to filter out less desirable components while the second rule selects a component that has a maximum degree of usage as quantified by $|f(n)|$.

To explain the dependency constraint, suppose component *A* depends on component *B*. Testing *A* before *B* would require the development of one stub to simulate *B* in order to test *A* while testing *B* before *A* would not require any stub. Therefore, *B* is preferred to *A* as a candidate for the next component to be tested. Thus, we can eliminate *A* from the candidate list. As shown in Figure 3, the algorithm uses this constraint to examine each pair of nodes *u* and *v* in a set of candidates *M.* If *u* depends on *v* but *v* does not depend on *u* then eliminate *u* since testing *u* before *v* would require more stubs than testing *v* before *u*. *P* collects all the nodes that are eliminated and so the algorithm only has to select the next component to be tested from the remaining nodes in *M* (except when the dependency constraint is not able to discriminate any node in *M* and collect all eliminated nodes in *P*). Next if there is more than one remaining nodes in *M*, the algorithm selects the first node found with a maximum degree of usage to be the next component in the test order. The degree of usage of a tested unit *n* can be estimated by all current components that need *n* for testing, i.e., $f(n)$, which is reflected by all components that depend on *n* but have not been tested yet. Thus, as shown in Figure 3, $f(n)$ contains nodes that are in $in(n)$ but not *T* and the cardinality of $f(n)$ gives an estimated measure of the degree of usage of *n*.

As shown in the procedure *Select* at the bottom of Figure 3, each time a candidate component is selected for integration testing, the algorithm appends the candidate to a test order *T*, maintains a set of required stubs *S*, and updates a set of components that remain to be considered for integration testing. The algorithm repeats until the sequence of test order covers all components in an integrated system. Note that removal of components, from a set of all components to be tested, does not require a new calculation of *in* and *out* in each iteration we compute $h(n)$ and $f(n)$ since the changes are already accounted for by using current *S* and *T* in the calculation.

4.2 Complexity Analysis

Finding test order to minimize the number of stub is NP-Complete [11]. The dependency constraint checking (for-loop in Figure 3) is the most time consuming step of the algorithm. However, in the worst case, for a graph with *n* components, it takes $O(n^2)$ time in the first (while-loop) iteration and no execution time in the rest because of the effect from our filtering mechanism. For example, consider a fully connected (complete) graph where there is a dependency between every pair of nodes. Initially, $|h(n)| = |f(n)| = n$ for every node. After the first iteration, a node is selected for testing in *T* and the rest of *n*-1 nodes are members of *S*. Therefore, in the next iteration $|h(n)| = 0$ for each of the rest of the nodes to be considered for integration testing. Thus, the first node found can be selected without requiring the dependency constraint checking. Therefore, our algorithm takes $O(n^2)$ time in the worst case. Comparing to other techniques that require identification of nested SCCs, each recursion takes $O(n^2)$ time and this gives an overall of $O(n^3)$ time in the worst case. However, our approach trades the time and complexity saved from checking if a node under consideration is a part of a cycle with the possibility of generating a solution for a class test order problem that may not be optimal. Our illustrations show that nevertheless, our algorithm performs competitively with other graph-based techniques while it is a lot simpler.

4.3 Specific vs. generic stubs

There are two types of stubs: *specific* and *generic*. A specific stub is developed for testing a specific dependent component, whereas a generic stub is for testing all dependent components. For example, class *C* contains methods *p* and *q*, where *A* calls *p* and *B* calls *q*. Therefore, *A* and *B* are dependent components of *C*. Testing them requires either one generic stub to simulate *C* or two specific stubs, one to simulate *p* (part of *C* relevant to test *A*) and the other to simulate *q* (part of *C* relevant to test *B*). Generic stubs are also called *realistic stubs* [11]. We adopt a new term to avoid further misperception of the implication that realistic stubs are the same as real implementations [2]. In fact, they

are not necessarily the same; for example, a generic stub can simply emulate I/O behaviors of a component.

Our algorithm as described in Section 4.1 deals with generic stubs. This is because once a stub is created for a certain component; any other dependent components of the same component can use the same stub. This exhibits generic stubs. To obtain specific stubs, our algorithm can be modified to obtain the test order and estimate the number of specific stubs by assuming that different classes that depend on the same class require different parts of this latter class. For example, classes *A* and *B* depend on different parts of class *C* and therefore testing them need two specific stubs of *C*. Other researchers [11] also apply this assumption to address the same issue. Consequently, our algorithm can deal with specific stubs by simply labeling each stub developed with its specific dependent class. Thus, our algorithm can be applied to both types of stubs in any level of details of the given OO model.

5. Illustrations

5.1 An example

For comparison purpose, we base our example on a relatively small example of the test dependency graph used in [11] but add a corresponding class diagram to give a comprehensive illustration of the proposed approach.

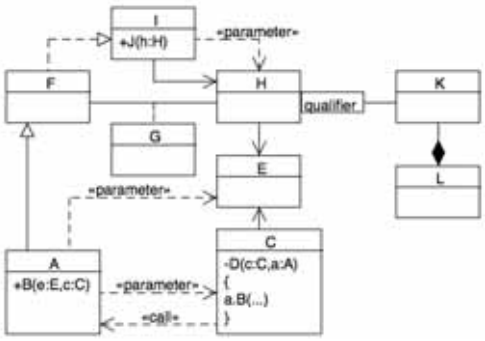


Fig 4 UML class diagram.

Figure 4 shows a class diagram of a given object-oriented software. Using our terminology on levels of OO models in Section 3, we apply appropriate mappings according to the levels of details in the models to construct a test dependency graph *G* in figure 5. For example, since class *A* has method *B* that has parameters of classes *C* and *E*, by applying a parameter dependency mapping in Figure 2, we obtain a corresponding test dependency that *A* depends on *B*, which depends on *C* and *E*.

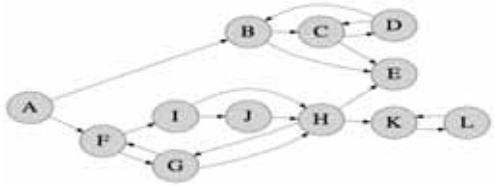


Fig 5 A test dependency graph example.

Now we apply the test order generation algorithm as described in Section 4.1. Table 1 shows the results of the first six iterations obtained by our algorithm using an input test dependency graph of Figure 5. The first three columns represent components, sets of *in* and *out* of each component in the given graph. The shaded area of Table 1 shows a set of stubs, *S* and a test order sequence generated so far, *T*. For simplicity, we omit commas and curly brackets in a set notation (e.g., *S, in,* and *out*). Similarly, we omit commas and angle brackets in a sequence notation for *T*.

Table 1 Application of the proposed algorithm.

Iteration:			1	2	3	4	5	6
S	Ø		Ø	*D*	*D*	*D*	*DL*	*DL*
T	*null*		*E*	*EC*	*ECB*	*ECBD*	*ECBDK*	*ECBDKH*
V	*out*	*in*	$\|h\|$	$\|h\|$ $\|f\|$	$\|h\|$	$\|h\|$	$\|h\|$ $\|f\|$	$\|h\|$
A	*BF*		2	2	2	1	[1] 1	1
B	*CE*	*AD*	2	✗	⓪			
C	*DE*	*BD*	2	[1] ②				
D	*BC*	*C*	2	2 **S**	1	⓪		
E	Ø	*BCH*	⓪					
F	*GI*	*AG*	2	2	2	2	2	2
G	*FH*	*F*	2	2	2	2	2	2
H	*EK*	*GIJ*	2	✗	1	1	✗	⓪
I	*HJ*	*F*	2	2	2	2	2	2
J	*H*	*I*	1	✗	1	1	✗	1
K	*L*	*HL*	1	[1] 2	1	1	[1] ②	
L	*K*	*K*	1	[1] 1	1	1	[1] **S**	0

Recall that $h(n)$ is a set of nodes that are in $out(n)$ but not in *S* or appears in *T*. Thus, in the first iteration, $|h|$ is the same as $|out|$. As shown in Table 1, select *E*, a component with minimal $|h|$ of zero value (indicated by a circled value) to be tested (as indicated by *T* in the first iteration). To compute stubs required for testing *T,* since $out(E)$ is empty (i.e., *E* does not depend on any component), no stub is required (as indicated by *S* in the first iteration). In the second iteration, first re-compute $|h|$, since *E* is in the *out* of *B, C,* and *H,* therefore $|h|$ of each of these nodes are reduced by one. The rest remains the same. There is more than one component with minimal $|h|$ (indicated by boxed values), namely *B, C, H, J, K* and *L*. The algorithm then uses a dependency constraint to eliminate nodes from these six candidates. Since *B* depends on *C* (i.e., $C \in out(B)$) but *C* does not depend on *B,* therefore by the dependency constraint, *B* is eliminated. Similarly, we eliminate *H* and *J* (as indicated by a cross on each of these candidate boxes). For the remaining candidates *C, K* and *L,* the algorithm selects the first component found with a maximal degree of usage measured by $|f|$. Recall that $f(n)$ is a set of nodes that are in $in(n)$ but not in *T*. Thus, we compute $|f|$ for *C, K, L* and obtain their values as 2, 2, and 1, respectively. As shown in Table 1, *C* is selected in the second iteration to be the next component in *T*. Since *C* depends on *D* and *E,* where *E* is in *T,* therefore *C* only requires a stub that simulates *D* (indicated by a symbol **S** on a corresponding component row). The algorithm repeats until *T* contains all components in the

graph. It terminates after 11 iterations with final results of T = <*E, C, B, D, K, L, H, J, I, F, G, A*> and S = {*D, L, G*}. The number of (generic) stubs obtained by our algorithm is the same as the minimal number of stubs obtained by Traon et al. [11] while the class test order and a set of stubs are slightly different.

5.2 Comparisons on a case study

This section applies our approach to a case study of a server design and implementation of a *switched multimegabit data service* (SMDS) in telecommunication. Due to limited space, we omit the details of the design descriptions and a corresponding UML model, which can be found in [5]. Figure 6 shows a test dependency graph of the UML model of the SMDS system as given in [11].

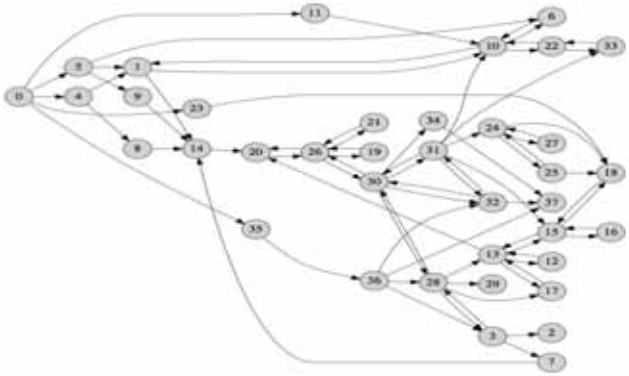


Fig 6 A test dependency graph of the SMDS OO model.

As shown in Figure 6, there are 38 components in the (cyclic) graph. Our algorithm terminates with a recommended test order set T and a set of generic stubs S as shown in the bottom part of Table 2.

Table 2 Comparisons with Traon et al.'s approach.

Traon et al.	Test order, T'	< 2, 29, 37, 34, 17, 12, 16, 18, 15, 27, 25, 24, 19, 21, 20, 13, 14, 7, 1, 6, 33, 22, 10, 31, 32, 3, 28, 30, 26, 8, 9, 36, 23, 11, 4, 5, 35, 0 >
	Set of stubs, S'	{10, 13, 15, 22, 24, 26, 28, 30, 32} (generic)
	# Stubs	9 (generic), and 20 (specific)
Hewett et al.	Test order, T	< 37, 2, 29, 34, 18, 16, 23, 20, 14, 7, 8, 9, 19, 21, 1, 6, 4, 5, 11, 3, 10, 33, 22, 15, 17, 12, 13, 28, 26, 32, 30, 36, 35, 0, 25, 27, 24, 31>
	Set of stubs, S	{10, 13, 15, 22, 24, 26, 28, 30, 31} (generic)
	# Stubs	9 (generic), and 20 (specific)

The top part of Table 2 shows the results obtained by Traon et al. [11]. Although the test orders obtained by both approaches are different, both require the same number of nine generic stubs, all of which simulate the same stub components except one. We also apply our algorithm for specific stubs and obtain the same number of required specific stubs as that of Traon et al. In this case, our approach appears to perform as well as Traon et al's technique, which claims to produce near optimal solution.

To compare our results with other graph-based techniques, we apply our approach to an example given in [2].

Table 3 Comparions with other graph-based approaches.

	Tai and Daniels	Traon el al.				Briand el al.	Hewett et al.
		Start with H			Start with G		
		Choose vertex A	Choose vertex E	Choose vertex F			
#stubs	5	3	3	3	3	4	3
# spec. stubs	5	7	6	6	4	4	5

Summary of results are shown in Table 3. In the case of generic stub, our approach and Traon el al.'s [11] outperformed the rest, whereas in case of specific stubs, Brian et al.'s and Traon et al.'s approaches outperform others although the latter has some advantages from non-determinisms during identification of SCCs and selecting root vertices for cycle breaking. Our approach only produces a near optimal solution for specific stubs. This is the subject of our future research.

6. Conclusion

We present an approach to a cost-effective testing strategy for incrementally integrating and testing object-oriented software that aims to minimize the number of stubs. Unlike existing approaches that focus on identifying ways to break cycles in a given test dependency graph, our approach directly choose components based on three simple heuristics.

Future work includes an extension to specifically address test order generation that minimizes specific stubs and further refinement by using different types of relationships in UML models and different types of stub development to guide test order generation.

References

[1] Binder, R., 1996. Testing object-oriented software: A survey, *J. Software Testing, Verification, Reliability,* 6: 125-252.

[2] Briand, L., Y. Labiche and Y. Wang, 2003. An investigation of Graph-based Class Integration Test Order Strategies, *IEEE Transactions on Software Engineering,* 29(7): 594-607.

[3] Cormen, T., C. Leiserson, R. Riviest and C. Stein, 2001. *Introduction to Algorithms,* McGraw Hill.

[4] Harrold, M., J. McGregor and K. Fitzpatrick, 1992. Incremental testing of object-oriented class structures, in *Proc. of Int'l Conf. of Software Engineering,* pp. 68-80.

[5] Jézéquel, J., 1996. *Object Oriented Software Engineering with Eiffel,* Addison-Wesley.

[6] Jorgensen, P. and C. Erikson, 1994. Object-oriented integration testing, *Communications ACM,* 37: 30-38.

[7] Kung, D., J. Gao and C. Chen, 1996. On regression testing of object-oriented programs, *J. Sys. and Software,* 32(1): 21-40.

[8] McGregor, J. and D. Sykes, 2001. *A Practical Guide to Testing Object-Oriented Software,* Addison-Wesley.

[9] Rumbaugh, J., J. Jacobson and G. Booch, 1998. *The Unified Modeling Language Reference Guide:* Addison-Wesley.

[10] Tai, K. and F. Daniels, 1999. Interclass test order for object-oriented software, *J. Object-Oriented Prog,* 12(4):18-25.

[11] Traon, Y. L., T. Jéron, J. Jézéquel and P. Morel, 2000. Efficient Object-Oriented Integration Testing and Regression Testing, *IEEE Transaction on Reliability,* 49(1): 12-25.

Using Observer Automata to Select Test Cases for Test Purposes

Gordon Fraser Martin Weiglhofer Franz Wotawa*
Institute for Software Technology – Graz University of Technology
{fraser,weiglhofer,wotawa}@ist.tugraz.at

Abstract

The use of formally specified test objectives, commonly known as test purposes, has lead to efficient test generation tools based on a well-defined theory in the domain of labeled transition systems. While a good test purpose can reduce the size of the state space that has to be considered to a manageable fragment, it can still result in an infinite number of test cases, out of which a single test case is selected in practice. Because writing test purposes is a difficult manual task, a single test case per test purpose might result in weak test suites. In this paper we present a technique that uses observer automata representing coverage criteria to select finite test suites for test purposes. The method is applied to an industrial application, and the effects on performance and the fault detection ability are measured.

1. Introduction

Software testing is an important but complex task. To aid the tester, model-based testing techniques use formal test models to automatically derive test cases and determine the outcome of the test execution. Assuming the existence of such a suitable model, the dreaded state explosion problem remains as one of the main issues in model-based testing. Formally specified test purposes have been successfully used to cut down the size of a model to manageable chunks [4, 5, 8]. There is a well-defined theory on test case generation in the domain of labeled transition systems [9], and efficient tools such as TGV [8] based on test purposes have been made available.

For a given test purpose, tools like TGV can derive either a single test case satisfying the test purpose, or a graph that subsumes all possible test cases. To create good test suites with the first approach, a set of suitable test purposes is required. Creating test purposes, however, is a non-trivial task. For example, du Bousquet et al. [2] report that even after ten hours of manual test purpose design they failed to find a set of test purposes that lead to detection of all mutants of a given implementation.

Creating several test cases for each test purpose therefore seems to be a feasible alternative. However, the generation of a graph that subsumes all test cases as provided by TGV can not be seen as a final step in the test case generation. Potentially, such a graph might represent an infinite number of test cases. Unfortunately, not every linear trace of the graph is a valid test case according to the theory of conformance testing (ioco) [9].

There is a whole spectrum of different possibilities for test case selection ranging from a single test case up to all possible test cases for a given test purpose. Surprisingly, the issue of whether and how to select several test cases for a test purpose has hardly been considered before. This paper aims to fill this gap. For this we use established test selection strategies for models based on coverage criteria. We extend observer automata based techniques [1, 6] to test case selection to Tretmans's input/output conformance testing theory [9], and show how this can be applied to select several test cases from a single test purpose.

The technique is evaluated using an industrial application. The corresponding specification is derived from informal standards, and TGV is used to create complete test graphs that represent the superset of all possible test cases for a test purpose. Observer automata for different coverage criteria are then used to extract finite sets of test cases from the test graphs, and resulting test suites are evaluated with regard to their coverage and fault detection ability.

The remainder of this paper is organized as follows: Section 2 describes the necessary preliminaries of conformance testing for labeled transition systems. Then, Section 3 describes how observer automata can be applied to select test cases according to coverage criteria from a given test graph. Section 4 presents the results of an empirical evaluation, and Section 5 concludes the paper.

2. Preliminaries

The results presented in this paper are based on Tretmans's theory [9] of conformance testing for input-output

*Authors are listed in alphabetical order.

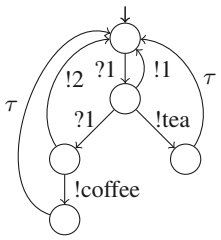


Figure 1. A labeled transition system.

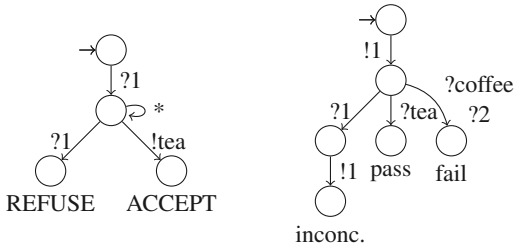


Figure 2. A test purpose and a test case.

labeled transition systems. Labeled transition systems are used to define the semantics of many common specification languages, e.g., LOTOS [7]. This section briefly summarizes the necessary concepts and preliminaries.

Definition 1 (Labeled Transition System) *A labeled transition system (LTS) is a tuple $M = (Q^M, A^M \cup \{\tau\}, \rightarrow_M, q_0^M)$, where Q^M a finite set of states, A^M a finite alphabet and $\tau \notin A^M$ is an unobservable action, $\rightarrow_M \subseteq Q^M \times A^M \times Q^M$ is the transition relation, and $q_0^M \in Q^M$ is the initial state.*

An LTS is *deterministic* if for any sequence of actions from the initial state there is at most one successor state. An LTS is *complete* if it allows all actions in every state. In order to properly test a system using an LTS it is necessary to distinguish the system's inputs and outputs:

Definition 2 (Input Output Labeled Transition System) *An input output labeled transition system (IOLTS) is an LTS $M = (Q^M, A^M \cup \{\tau\}, \rightarrow_M, q_0^M)$ where A^M is partitioned into two disjoint sets $A^M = A_I^M \cup A_O^M$, where A_I^M and A_O^M are input and output alphabets, respectively.*

Figure 1 shows an example IOLTS representing a drink vending machine. The labels of input actions have the prefix "?", and output actions have the prefix "!". After receiving a coin from the user, the system can return tea, reject the coin, or receive another coin, in which case coin rejection or coffee as output are possible.

Because the state explosion problem often makes test case generation using simple coverage criteria on the whole model impossible or difficult, the idea of *test purposes* is to represent a test objective in a way that significantly cuts down the size of the state space that needs to be considered. A test purpose is a formal definition of a test objective, and used by tools like TGV for automated test case generation. TGV defines a test purposes as follows [8]:

Definition 3 (Test Purpose) *A test purpose is a complete, deterministic IOLTS $TP = (Q^{TP}, A^{TP}, \rightarrow_{TP}, q_0^{TP})$, equipped with two sets of trap states $Accept^{TP}$ and $Refuse^{TP}$, with the same alphabet as the specification S, i.e. $A^{TP} = A^S$. A trap state q has a self-loop for each action, i.e. $\forall a \in A^{TP} : q \xrightarrow{a}_{TP} q$.*

An example test purpose is shown on the left hand side of Figure 2. This test purpose aims to select such test cases where the output is tea. Any behavior after inserting two coins successively is refused, which means that this behavior is not subject of this test purpose. The $*$-labeled edge selects any edge, except those labeled with either $?1$ or $!tea$.

Given a test purpose and a formal specification, TGV generates either a single test case or a complete test graph, which contains all test cases corresponding to the test purpose. The reduction of the state space depends on the use of reject states in the test purpose. Except for controllability a test graph already satisfies the properties of a test case.

Definition 4 (Test Case) *A test case is a deterministic IOLTS $TC = \left(Q^{TC}, A^{TC}, \rightarrow_{TC}, q_0^{TC}\right)$ equipped with three sets of trap states $Pass \subset Q^{TC}$, $Fail \subset Q^{TC}$, and $Inconclusive \subset Q^{TC}$ characterizing verdicts. A test case has to satisfy the following properties: (1) TC mirrors image of actions and considers all possible outputs of the IUT; (2) From each state a verdict must be reachable; (3) States in $Fail$ and $Inconclusive$ are only directly reachable by inputs; (4) A test case is input complete in all states where an input is possible; and (5) TC is controllable, i.e., no choice between two outputs or between inputs and outputs.*

The right hand side of Figure 2 shows a test case corresponding to the test purpose to its left. As can be seen, outputs of the specification are inputs for the test case and vice versa. The test case gives a pass verdict if the output of tea is observed after a coin is inserted. The test case ends in an inconclusive state if the first inserted coin is rejected.

In the succeeding section observer automata for coverage criteria are introduced as a compromise between a single test case per test purpose and the complete test graph. Because observer automata add conditions to transitions, we also need to define symbolic transition systems (STS) [3]:

Definition 5 (Symbolic transition system) *A symbolic transition system is a tuple $\mathcal{S} = (L, l_0, \mathcal{V}, \mathcal{I}, \Lambda, \rightarrow)$, where L is a set of locations and $l_0 \in L$ is the initial location. $\mathcal{V}$ is a set of location variables and $\mathcal{I}$ is a set of interaction*

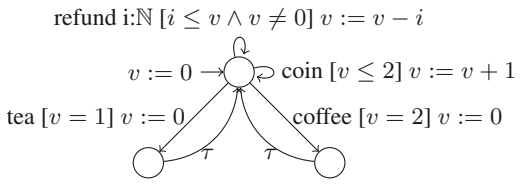


Figure 3. A symbolic transition system.

variables with $\mathcal{V} \cap \mathcal{I} = \emptyset$. Λ is the set of actions, and $\tau \notin \Lambda$ denotes an unobservable action. $\rightarrow$ denotes the transition relation, where each element $(l, \lambda, \phi, \rho, l') \in \rightarrow$, has a source location l, a target location l', a possibly parameterized action λ, a guard ϕ and a update function ρ.

Figure 3 shows an example STS, again representing a drink vending machine. Transitions are labeled with an action (refund, tea, coin, coffee), guard conditions are included in brackets [] and followed by the update function. The variable v stores the number of inserted coins. The action refund uses an additional parameter i, which is a natural number. Due to the guard this parameter is less than or equal to the number of inserted coins, i.e., $i \leq v$. Thus, this action allows the rejection of the inserted money.

Although symbolic transition systems extend LTSs by incorporating data and data-dependent control flow, the semantics of STS is given by an corresponding LTS. For a detailed discussion of STSs we refer to [3].

3. Observer-based Test Case Selection

The use of observer automata for generating test cases with respect to specific coverage criteria has been proposed by Blom et al. [1]. Observer automata can be created for many different coverage criteria. They are used during state space exploration in order to detect when coverage items have been reached and return corresponding test cases. Informally, the superposition of an observer automaton and the model is calculated and traversed. Whenever an observer enters an accepting state the linear trace that lead to this state is a test case that covers the coverage item represented by the observer.

Figure 4 depicts an observer automaton as an STS. The observer stays in its initial state as long as the desired coverage item is not reached. We denote the coverage item as "cov.item", and it can represent any entity of the LTS that should be covered, for example states, labels, or transitions. Note, that symbolic transition systems do not provide accepting states as required by observer automata. Formally, we use trap states, i.e., states with self-loops for each action, to implement accepting states.

In our formal setting, an observer automaton is a symbolic transition system that is parametrized by a particular

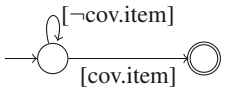


Figure 4. Coverage observer automaton.

coverage item. In addition, the ioco-theory requires additional properties for test cases (Definition 4) that cannot always be fulfilled by linear traces. Consequently, a test case is an IOLTS which has to be input complete in states where inputs are possible and controllable. In addition, the definition of a test case requires that only responses from the system under test lead to the verdicts fail and inconclusive, and that from each state a verdict must be reachable. Finally, outputs of the system under test have to be inputs for the test case and vice versa.

Except for controllability, a test graph generated by TGV fulfills all the required properties. Thus, any approach that extracts test cases from such a test graph needs to preserve these properties, while it must additionally ensure controllability. Below we show how to extend the approach based on observer automata in order to generate ioco-correct test cases.

3.1. Extending Observer Automata

In its original definition [1], an observer automaton generates a linear test case that ends as soon as the coverage item has been covered. Such test cases do not satisfy the requirements of ioco test cases.

Thus, observer automata have to ensure that every trace of a generated test case ends in a verdict state. Because test cases have to be input complete (Definition 4) they might include traces that do not cover the observer's coverage item. Such traces result from inputs that have to be selected in order to satisfy input completeness. An additional variable *allows* is used to take care of traces leading to verdict states but not covering the coverage item of the observer. Such traces are only allowed to be in the test case if the coverage item has been covered by another trace selected by the observer.

This extension is illustrated in Figure 5. Any observer automaton for a particular coverage item (cov.item) can be extended in this way. Let $V = Pass \cup Fail \cup Inconclusive$, then for an edge (s, a, s') of the superposition of the observer automaton and the model this extension distinguishes three different cases:

1. The edge leads to a verdict state and the coverage item is met by that edge: $s' \in V \wedge \neg cov.item$

2. The edge meets the coverage item but it does not lead to a verdict state: $s' \notin V \wedge cov.item$

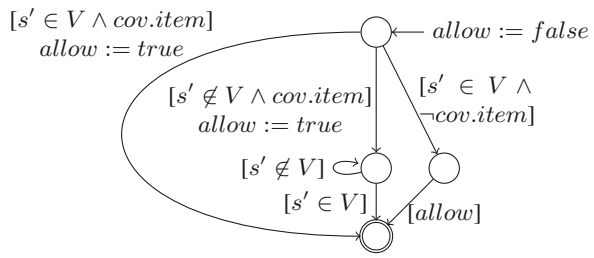


Figure 5. Extended Observer for the generation of ioco-correct test cases.

3. The edge leads to an verdict state, but the edge is not related to the coverage item: $s' \in V \wedge \neg cov.item$

3.2. Ioco Based Test Case Extraction

Figure 6 depicts the proposed algorithm that deals with the discussed issues. Similar to [6], we use two data structures *wait* and *pass* for maintaining states waiting to be examined and states already examined. The set $wait$ consists of triples P and the test case ω associated with these triples. Each triple comprises the state of the test graph s, the states of the observers C and the corresponding state of the generated test case t.

As long as there are elements in $wait$ (Line 2), the algorithm takes pairs (P, ω) from $wait$. Then it iterates over all triples in P (Line 5) and considers the successor states $\langle s, C\rangle$ given by the transition relation $\rightarrow_C$ of the observer/test graph superposition (Line 6 and 20). If a successor state is reached by an output action (Line 6) and if this state has not been processed previously (Line 7) then the current set of triples P is copied to P'. Note, that "_" in Line 7 and in Line 25 matches any test case state. The selected triple $\langle s, C, t\rangle$ is replaced within P' by a new triple built from the successor $\langle s', C'\rangle$ of $\langle s, C\rangle$ and a new state t_{new} within the partially built test case ω. The test case ω is updated by adding the considered transition to ω. Note, that due to the use of a new unused state of ω we implicitly unroll loops of the test graph. If all triples correspond to final states in the observer (Line 10), then the test case is added to the test suite, otherwise the new P' is added to $wait$.

If a successor is connected via an input edge (Line 20), then the successor triple is added to a temporary datastructure P' that will hold all successor triples reachable via inputs. After iterating over all successors (Line 20) the test case is either added to the test suite (Line 27) or the newly generated state is added to the waiting states.

Optimization Note, that the algorithm illustrated in Figure 6 allows to optimize the number of generated test cases.

```
1:  passed ← ∅; wait ← {(⟨s0, C0, t0⟩, ϵ)}
2:  while wait ≠ ∅ do
3:      select (P, ω) from wait
4:      add (P, ω) to pass
5:      for all ⟨s, C, t⟩ ∈ P do
6:          for all output edges e : ⟨s, C⟩ ⟹^a_C ⟨s', C'⟩ do
7:              if ⟨s', C', _⟩ ∉ (pass ∪ wait) then
8:                  t_new ← new (unused) state in ω
9:                  ω' ← ω' ∪ (t, b, t_new)
10:                 if new covered observer in C' then
11:                     add ω to test suite
12:                 else
13:                     P' ← P\⟨s, C, t⟩ ∪ ⟨s', C', t_new⟩
14:                     add (P', ω') to wait
15:                 end if
16:             end if
17:         end for
18:         if there are input edges then
19:             (P', ω') ← (P, ω)
20:             for all input edges ⟨s, C⟩ ⟹^b_C ⟨s', C'⟩ do
21:                 t_new ← new (unused) state in ω
22:                 P' ← P'\⟨s, C, t⟩ ∪ ⟨s', C', t_new⟩
23:                 ω' ← ω' ∪ (t, b, t_new)
24:             end for
25:             if ⟨s', C', _⟩ ∉ (pass ∪ wait) then
26:                 if new covered observer in C' then
27:                     add ω' to test suite
28:                 else
29:                     add (P', ω') to wait
30:                 end if
31:             end if
32:         end if
33:     end for
34: end while
```

Figure 6. Extended observer automata based test case extraction algorithm.

We can skip test cases that do not cover new observers. Basically, we are interested in a set of test cases such that every observer is covered. Similar to [6] we approximate this set by adding a test case (Lines 11 and 27) only to the test suite if it covers a currently uncovered observer.

4. Empirical Evaluation

In order to show the improvements gained from extracting multiple test cases for a single test purpose we applied the presented techniques to the Session Initiation Protocol (SIP). This section presents the results in terms of mutation score and source code coverage in comparison to the results of the single test case selection strategy. Note that it is not feasible to select all test cases in a complete test graph in general, as the number of test cases can be infinite.

4.1. Coverage Criteria for Labeled Transition Systems

Observer automata can represent various coverage criteria. For our evaluation we used three different coverage criteria on the complete test graph: state coverage, label coverage, and transition coverage.

Definition 6 (State Coverage) *A state $q \in Q^M$ of a labeled transition system $M = (Q^M, A^M \cup \{\tau\}, \rightarrow_M, q_0^M)$ is covered by test case $t = (Q^t, A^t \cup \{\tau\}, \rightarrow_t, q_0^t)$, if $q \in Q^t$.*

The state coverage value is calculated as the ratio of covered states to states in total in the LTS.

Definition 7 (Label Coverage) *A label $l \in A^M$ of a labeled transition system $M = (Q^M, A^M \cup \{\tau\}, \rightarrow_M, q_0^M)$ is covered by test case $t = (Q^t, A^t \cup \{\tau\}, \rightarrow_t, q_0^t)$, if $l \in A^t$ and there exists a transition $(q_1, l, q_2) \in \rightarrow_t$.*

A label l of labeled transition system is covered by a test case if there is at least one transition of the test case labeled with l. The label coverage represents the percentage of labels of the LTS that are covered.

Definition 8 (Transition Coverage) *A transition $(q, l, q') \in \rightarrow_M$ of a labeled transition system $M = (Q^M, A^M \cup \{\tau\}, \rightarrow_M, q_0^M)$ is covered by test case $t = (Q^t, A^t \cup \{\tau\}, \rightarrow_t, q_0^t)$, if $(q, l, q') \in \rightarrow_t$.*

The transition coverage represents the percentage of transitions of the LTS that are covered.

Note, that in the case of a complete test graph, transition coverage subsumes label coverage and state coverage. This is because there are no states unreachable from the initial state. Furthermore, there are no labels that are never used on any transition. Note further, that label coverage does not subsume state coverage, since the same label my be used on different transitions.

4.2. The SIP Registrar Application

The Session Initiation Protocol (SIP) handles communication sessions between two end points. SIP defines various entities that are used within a SIP network. One of these entities is the *Registrar*, which is responsible for maintaining location information of users.

In cooperation with our industry partner's domain experts we developed a formal specification covering the full functionality of a SIP Registrar. This obtained LOTOS specification comprises approx. 3KLOC (net.), 20 data types (contributing to net. 2.5KLOC), and 10 processes. Details about our SIP Registrar specification can be found in [10].

4.3. Experimental Results

Table 1 shows the results in terms of the numbers of test cases generated with the discussed methods: A single test case per test purpose, state coverage (S), label coverage (L), and transition coverage (T). Five manually specified test purposes were used for this experiment. Table 2 illustrates the performance of our prototype implementation of the presented algorithm on a PC with Intel(R) Dual Core Processor 1.83GHz and 2GB RAM.

In particular, Table 2 shows for each manually generated test purpose the time needed to derive the number of test cases stated in Table 1. The left part of these tables shows the results using all test cases generated by the observer based approach, i.e., in that case we do not check for already covered observers. In contrast, the right part shows the results when only using test cases that cover new observers as described in Section 3.

Minimizing the generated test suite with respect to the covered observers allows to reduce the number of test cases in the state coverage based test suite by 44%. For label and transition coverage we get a reduction of approximately 4% and 12%, respectively. However, the minimization slows down the test case extraction.

Using multiple test cases per test purpose, the complete test graphs generated by TGV are split into single test cases within reasonable time. Note, that the time needed by TGV for generating a complete test graph is higher than the time needed to generate a single test case. That is, TGV needs approximately 3'48", 5", 19", 5", and 6", respectively, for the five test purposes. These times are not included in the figures listed in Table 2.

To assess the quality of the generated test cases Table 3 shows the mutation scores on mutated versions of the LOTOS specification. For this, we generated 633 mutants exhibiting observable faults, i.e. faults that can be detected using test cases derived with respect to input-output conformance. In addition, this table illustrates the function coverage and condition/decision coverage on the OpenSER implementation[1] of the SIP Registrar. Finally, Table 3 depicts the number of actual faults detected in OpenSER by the different test suites. The results listed in this table apply to the full test suites as well as to the minimized test suites; i.e, there was no observable degradation of the fault sensitivity through the minimization.

The mutation score, which is the percentage of mutants detected by the test cases, shows the improvement gained from using multiple test cases per test purpose. Using of a single test case per test purpose kills 45% of the 633 mutants, which is better than we expected.

The increase of source code coverage in terms of function coverage and condition/decision coverage is not as sig-

[1] http://www.openser.org

Table 1. Number of test cases.

TP	Sin-	Regular			Minimized		
	gle	S	L	T	S	L	T
1	1	33	324	2174	18	302	2098
2	1	12	1132	1145	11	1116	1116
3	1	507	1752	2764	280	1593	1943
4	1	2	662	662	1	660	660
5	1	4	999	1002	3	996	996
Σ	5	558	4869	7747	313	4667	6813

Table 2. Creation times.

TP	Sin-	Regular			Minimized		
	gle	S	L	T	S	L	T
1	24”	8”	19”	16’13”	8”	23”	19’41”
2	5”	<1”	1’20”	1’20”	<1”	2’01”	2’00”
3	3”	56”	5’20”	1’27”	54”	6’15”	11’29”
4	6”	<1”	28”	27”	<1”	41”	41”
5	5”	<1”	1’00”	1’00”	<1”	1’33”	1’33”
Σ	43”	1’07”	8’27”	20’27”	1’05”	10’53”	35’24”

nificant as the mutation score. This is because the OpenSER Registrar is implemented highly modularly and reuses large pieces of the registration message handling code.

5. Conclusions

While formally defined test purposes can be used to efficiently cut down the size of the state space that needs to be considered for testing, it is common practice to select single test cases per test purpose. As writing test purpose is a non-trivial task, we presented a method that increases the number of test cases created for a test purpose while keeping the number of test cases within realistic bounds. The selection strategy is based on coverage criteria, which are implemented as observer automata. We extend observer automata to ioco theory, and show how such observers are used in the context of complete test graphs. The results of an empirical evaluation on an industrial application demonstrate that the fault sensitivity of resulting test suites is improved. Future research will include performance improvements and more complex coverage criteria suitable for STS like, for example, data flow coverage criteria.

Although our approach improves the fault detection ability of a single test purpose, it does not remove the difficulty of writing good test purposes. This is because one may write test purposes where TGV runs out of memory and thus does not generate complete test graph. Obviously, for such test purposes it is not possible to generate test cases.

Acknowledgments The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

Table 3. Test suite evaluation.

	Single	State	Label	Transition
Mutation Score	45%	76%	80%	81%
Function Cov.	64%	70%	73%	73%
C/D Coverage	26%	31%	34%	34%
Detected Faults	2	5	5	6

References

[1] J. Blom, A. Hessel, B. Jonsson, and P. Pettersson. Specifying and generating test cases using observer automata. In *4th International Workshop on Formal Approaches to Software Testing*, volume 3395 of *LNCS*, pages 125–139, 2004.

[2] L. du Bousquet, S. Ramangalahy, S. Simon, C. Viho, A. Belinfante, and R. G. de Vries. Formal test automation: The conference protocol with TGV/TORX. In *Proceedings of 13th International Conference on Testing Communicating Systems*, pages 221–228, Dordrecht, August 2000.

[3] L. Frantzen, J. Tretmans, and T. A. C. Willemse. Test generation based on symbolic specifications. In *4th International Workshop on Formal Approaches to Software Testing*, volume 3395 of *LNCS*, pages 1–15. Springer, 2004.

[4] J. Grabowski, D. Hogrefe, and R. Nahm. Test case generation with test purpose specification by MSC’s. In *Proceedings of the 6th SDL Forum*, pages 253–266. Elsevier, 1993.

[5] W. Grieskamp, N. Tillmann, C. Campbell, W. Schulte, and M. Veanes. Action machines — towards a framework for model composition, exploration and conformance testing based on symbolic computation. In *Proceedings of the Int. Conference on Software Quality*, pages 72–79, 2005.

[6] A. Hessel and P. Pettersson. A global algorithm for model-based test suite generation. *Electronic Notes in Theoretical Computer Science*, 190(2):47–59, August 2007.

[7] ISO 8807: Information processing systems – open systems interconnection – LOTOS – a formal description technique based on the temporal ordering of observational behaviour, 1989.

[8] C. Jard and T. Jéron. TGV: theory, principles and algorithms. *Int. Journal on Software Tools for Technology Transfer*, 7(4):297–315, August 2005.

[9] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software - Concepts and Tools*, 17(3):103–120, 1996.

[10] M. Weiglhofer. A LOTOS formalization of SIP. Technical Report SNA-TR-2006-1P1, Competence Network Softnet Austria, Graz, Austria, December 2006.

Building Testable Components – A Systematic Approach and its Experimental Study

Jerry Gao, Wrihang Roberto Liang, Radhika Chhabra, and Ramyashree Swamyo
San Jose State University, Email: jerrygao@email.sjsu.edu

Ma Xiang
Huawei Technologies Co. Ltd.

Abstract

Component-based software engineering is becoming a popular approach to build software application systems due to its reduction in project development cost and time. One of major challenges is how to increase component testability to facilitate component validation by users and application engineers. Although there are a number of papers addressing this problem, engineers still lack systematic methods to create testable components using a systematic solution. In addition, there is a need of a reusable test bed to support testable components. This paper presents a systematic solution to generate testable components, and a reusable test platform to support testing of testable components. The solution includes: a component model and interface for testing, and approach to form testable components. In addition, a component test framework and a reusable test platform are reported to support testable components. Moreover, some experimental results are discussed to show its strong potential usefulness.

Keywords: Component-based software engineering, component testability, design for software testability, component testing, and software testing.

1. Introduction

Today component engineering is gaining substantial interest in the software-engineering community due to its advantage in cost reduction. To assure the quality of component-based system, how to create reliable components and perform component testing is very important [1][4].In the practice of component engineering, we have encountered some new problems and challenges in testing of software components and component-based software[1][2][3][4]. One of them is how to increase component testability [2][3][4].

Since 1990, there have been a number of papers discussing component testability and design for component testability from different perspectives. The first paper about component testability is written by R. S. Freendman in [6]. He defines his ***domain testability*** for software components as a combination of component ***observability*** and ***controllability***. In his definition, "***observability*** is the ease of determining if specific inputs affect the outputs - related to the undeclared variables that must be avoided". And "***controllability*** is the ease of producing specific output from specific inputs - related to the effective coverage of the declared outputs domain from the input domain". Later, R. V. Binder in [5] discusses software testability for object-oriented programs concerning object-oriented features. According to [4], component testability is two-fold. First, it refers to the degree to which a component is constructed to facilitate the establishment of component test criteria and the performance of tests to determine whether those criteria have been met. Second, it refers to the degree to which testable and measurable component requirements are clearly given to allow the establishment of test criteria and performance of tests. Hence, it is important to study component development methods, guidelines and standards that construct testable and measurable software components.

As pointed in [4], it is essential for component developers to construct deployable, executable, testable and manageable software components to reduce the test costs and efforts of diverse software components. In recent years, a number of published papers have focused on how to increase component testability by building testable or self-testable components [3][7] [8] [9][10] [11] [12] [13]. Most of them only focus on self-contained components. Moreover, we lack published papers addressing the reusable test platforms (or beds) for testable components. Furthermore, no published papers report case study and experimental results as well as experience and lessons on constructing and testing testable components in terms of development and testing costs.

This paper focuses on building testable software components and their reusable test platform for test automation, and it discusses a component architecture reference model and a consistent test interface for testable components as well as a supporting test platform. The major contribution of the paper is its systematic approach to generate testable components and a reusable test bed to support component test execution by reducing the costs and efforts in constructing component test harness (such as test driver and stub generation test execution). Furthermore, the paper reports our experimental results on building and validating testable components in the developed test bed. The comparison result suggests that the proposed approach has a strong potential to achieve black-box component test automation in a plug-in-and-test approach.

The paper is structured as follows. Section 2 reviews the basic concepts of testable component, including basic requirements, properties, and benefits. Section 3 discusses the related work on design for testable components. Section 4 presents a systematic approach to construct testable components and a supporting test bed for component test automation. Section 5 reports our case study results and test automation experience. Finally, the conclusions and future remarks are discussed in Section 6.

2. Basic Concepts of Testable Components

What is a testable component?

A *testable component* refers to a software component which is developed using a well-defined component test model, including standard test interfaces, test information formats, and required built-in solutions to support: a) regulated testing interactions between the component and its supporting test tools, b) test operations of a component's users and testers. A ***testable component*** must be deployable, executable, and testable in a given component test bed (or a test platform) which is compliant with its component test model. It should be constructed in a way to facilitate component testing and automation to reduce the validation cost of component testers and users. Unlike normal components, testable components must be constructed using a well-defined component architecture model and consistent test

interface. In addition, they have the following basic requirements and features.

- ***Requirement #1:*** A testable component should be deployable and executable.
- ***Requirement #2:*** A testable component must be traceable by supporting a basic component tracking capability so that it enables a user (or a tester) to monitor component black-box test behaviors. As defined in [8], traceable components are ones constructed with a built-in tracking mechanism for monitoring various component behaviors in a systematic manner.
- ***Requirement #3:*** A testable component must provide a consistent, well-defined and built-in interface, called the test interface, to support external interactions for software testing. Although different components include diverse functional interfaces, they must include a consistent test interface to support software testing. This is very important to automate component testing, and reduce test costs on environment setting and test driver construction.
- ***Requirement #4:*** A testable component must include some program code that facilitates component testing by interacting with external testing facilities or tools to support test set-up, test execution and test validation.

Software components in a component-based system can be classified into two groups based on its dependency on other components:

- *Independent component (IC)*: An ***independent component (IC)*** refers to a component which has no dependency to other components. In other word, an IC component is a self-contained component which can be deployed and executed independently in a test environment or a user's targeting environment.

- *Non-independent component (NIC)*: A ***non-independent component (NIC***) refers to a component which depends on other components by accessing their API. In other word, a NIC component is not a self-contained component which cannot be executed and deployed without the presence of its dependent components. A NIC component has its component interaction interfaces with one or more interaction ports to other components.

Why do we need testable components?

The major objective to introducing the concept of testable components is to find a new way to develop software components with good testability so that they are easy to be executed, traced, observed, controlled, and tested. Using testable components enables to component testability to be enhanced and achieve component test automation in the following aspects:

- Convert components (including COTS components) to testable components easily.
- Standardize the test interface for components so that various test tools and facilities can be deployed and used easily. For example, a standardized component test interface simplifies diverse component interactions with a test management system and a component test execution tool.
- Use a systematic approach to automate the generation of component test drivers and stubs.
- Reduce the effort and cost of setting up component test beds to enable component test automation is a plug-in-and-test manner.

How to construct a testable component?

According to [4], there are three approaches to increase component testability:

- Method #1: Framework-based testing facility – In this approach, a well-defined framework (such as a class library) is developed and used to allow engineers to add program test-support code into components according to a provided application interface of a component test framework.
- Method #2: Built-in tests – In this approach, test-support code and built-in tests are added inside a software component as its parts to make it testable and support self-tests.
- Method #3: Systematic component wrapping for testing – In this approach, a systematic way is used to convert a software component into a testable component (or generate testable component) by wrapping it with a standard software component test interface and its supporting program code to support component unit testing.

A detailed high-level comparison is presented in [8].

3. Related Work

Recently, there are a number of papers addressing the design for component testability. They can be classified into three approaches. The first is known as built-in tests components. Dr. Wang et al in [3][9] proposes an approach to constructing built-in test (BIT) components for maintainable software. In this approach, built-in tests for a component are built as the parts of software components. Based on the built-in tests, a component operates in two modes: a normal mode and a maintenance mode. In the normal mode, a component operates its application functions. In the maintenance mode, the built-in tests of this component can be activated to validate component functions. Elaine Martins et al. [14] also use a similar idea to construct self-test components by adding BIT tests inside object-oriented software components written in C++. They use a consistent approach to integrating assertions and result checking codes into C++ classes as built-in-tests. A specific test driver is provided to support the activations of BIT components. Similarly, Le Treon et al. in [13] present a pragmatic approach for linking design and test of classes, seen as basic unit test components. Components are self-testable by enhancing them with embedded test sequences and test oracles. Self-testable components serve as building blocks for performing systematic integration and non-regression testing. In addition, Ram Chillarege [12] presents a way to insert probes into component source codes to detect the expected faults inside components to support self-tests.

The second approach is BIT wrappers for component testing proposed by Stephen Edwards in [10][11]]. He utilizes a model-based specification language called RESOLVE. His architecture builds on current research in systematically detecting interface violations in component-based software. He suggests each component provide a simple "hook" interface (with no run-time overhead) that can be used in adorning the component with BIT capability like self-checking and self-testing.

The third approach is known as testable beans. In [8], Jerry Gao et al. introduce a new concept of testable beans – testable components, which are designed to facilitate component testing. Unlike the previous approaches, they use a framework-based approach to construct testable beans based on a well-defined common test interface for support component testing. Each testable bean consists of the following parts: a) a component test interface supporting test operations, b) built-in test code supporting the interactions between component APIs and the test interface, c)

component tracking interface for monitoring component operations and behaviors, and d) built-in tracking code for component tracking. In [7], Jerry Gao et al. also discuss a framework-based method to construct traceable components to facilitate the monitoring of component behaviors for component-based systems.

The research work reported in this paper is an extension of our previous work in [7][8]. It is also influenced by Stephen Edward's BIT wrappers. Unlike the existing work, this paper uses a systematic method to construct a testable component without embedding component tests and probes inside components. Instead, it uses a well-defined component architecture model with a standard component test interface and a component test framework to support component test automation.

Unlike the previous work, this paper uses a systematic wrapping approach based on a common component test framework. In this approach, each component is required to provide its well-defined API and interaction descriptions in a component interface description language (known as CIDL). Based on the given CIDL descriptions, a component test wrapper can be automatically generated to bridge a standard component test interface and its application function interface. For any given component with CIDL descriptions, it can be converted into a testable component by combing its test wrapper, test interface and its original black-box component. Unlike BIT-components, testable components in this paper do not contain any component tests. All component tests (or test scripts) can be stored in a test repository. They can be activated using the provided component test bed to support their execution. Unlike the existing unit test tool, this test bed can be used to support various testable components with standard test interfaces and CIDL descriptions. Furthermore, the proposed approach can be used to deal with non-independent components by providing automatic generated component test connectors to support the interactions with its dependent components as a consistent handle for component test stubs. Using this approach we can convert given a COTS component to a testable component using automatic generated component wrappers. Here component wrappers played as pluggable component test harnesses for components to support component test automation.

4. Constructing Testable Components

This section presents a systematic approach to constructing testable components, or converting given components into testable components. It consists of four parts: a) a proposed architecture model for testable components with a standard test interface, b) a component test framework, c) systematic component test adaptor generation, and d) a consistent component test bed.

a) The Proposed Architecture Model for Testable Components

Figure 5 shows our proposed architecture for testable components. The idea is an extension of the previous work [7][8] to make testable component as a plug-in-and-test component bed. Here we assume software components are reusable components with specified deliverables in. Figure 1(b). In this architecture model, a testable component consists of the following additional parts for a given software component.

- A well-defined standard component test interface – This is an external component test interface, which is used as a standard test interface to interact with component test and management tools, such as test execution tool, and test management tool. This test interface is only useful for component unit testing. The advantage of using this standard component test interface is to cope with diverse component application interfaces. To support this interface, an internal test adaptor is needed to support the interactions between the built-in component test interface and the component application interface. The detailed description is given below.
- A component test wrapper, which is (or statically) generated dynamically based on component API specifications. It plays as an adaptor to facilitate the interactions between the standard component test interface and its application interface of each under test component.

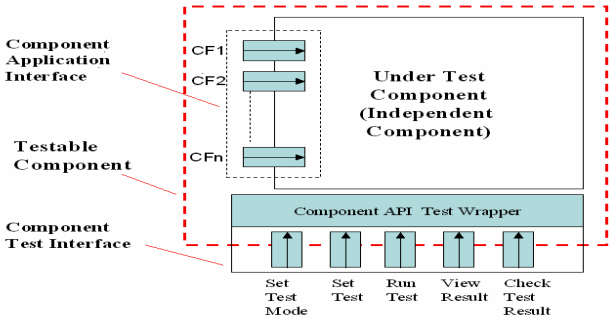


Figure 4 Component Architecture Reference Model

Component API Test Interface

As shown in Figure 4, CF1,.., to CFi correspond to the functional application interface of a given component. A standard component test interface is defined to allow component public users (its users and testers) to use it to exercise a given test case (or script) and view its test results. The basic features of a standard component test interface include the followings:

- It supports the common and standard test interface functions and operations.
- It is a reusable abstract interface that can be derived, instantiated or implemented with the support of actual component API wrapper.
- It must be supported by a component API test wrapper to bridge to its provided component application interface functions.
- It should be independent from a component's detailed application functions.

Interface
-m_nTEST_CompMode : int -m_pTestCase : TestCase -m_pTrackCase : TestCase -m_pNormalCase : TestCase
+CF1() +CF2() +CFn() +InitWrapper() : int +SetMode(in iMode : int) : int +SetUpTest(in argc : int, in argv : char**) : int +RunTest() : int +ValidateTestResult() : int +GetTestResult() : char* +RunComponent(in argc : int, in argv : char**) : int +GetInf() : char* +GetComponentFunction() : char* +Quit() : int

Figure 5 A Sample of Component Test Interface

As shown in Figure 5, a sample component test interface includes the following basic functions:

- Set mode (SetMode) – which sets the under-test component into a test mode or a normal execution mode.
- Set up test (SetUpTest) – which sets up a given API-based test case (including its inputs and expect outputs).
- Run test (RunTest) – which executes a component test case.

- Get Test Results (GetTestResult) – which collects the test results from an executed test.
- Validate Test Result (ValidateTestResult) – which checks a component's execution result against the expected test result in a given test case.
- GetComponentFunction(…) retrieves the component API-based functions and their signatures from a component API description
- GetInf(…) retrieves the test data for a given component test.

The major benefits using a standard component test interface is listed below.

- Establish a standard component interface to support test operations.
- Increase component testability by providing component test controllability.
- Reduce the component test harness cost in creating component test drivers and stubs.
- Provide a fundamental base for component test automation
- Standardize and simplify the component interactions and interfaces with software test tools.

Component API Test Wrapper and Its Generation

What is a component API test wrapper? It is a reusable part embedded in a test component to increase a component's controllability. It plays an adaptor of a component to interact with a standard component test interface, and maps a specific component API to the defined standard component test interface. It can be generated systematically in our proposed solution. The major function of a component API test wrapper is to set up the required entities supporting to bridge a under test component to the standard component API test interface.

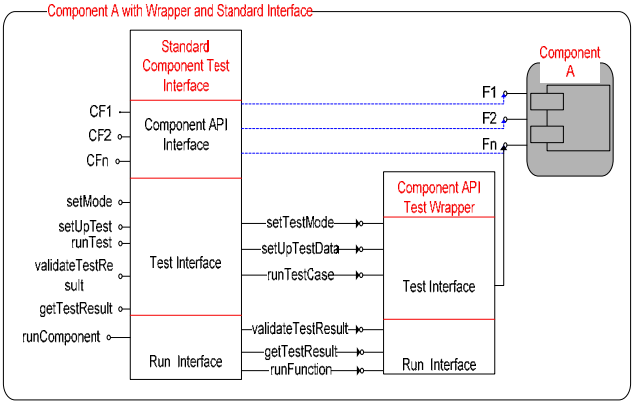


Figure 6 the Structure of a Testable Independent Component

Figure 6 shows the relationships between a component API test wrapper and a component test interface. In our approach, we use a systematic way to create a component API test wrapper for each component based on its consistent API specifications, written in the Component Interface Description Language (CIDL). For detailed specifications about CIDL and the detailed implementations can be found in [18]. Due to the limited space, we only explain the idea briefly. The CIDL includes three parts: a) *component profile specifications*, b) component API specifications, and c) component interaction specifications.

C*omponent profile specifications* provide a component's name and ID, version and release, properties (such as platform and language), and configurations. The high-level syntax is shown below.

Component-Profile: Comp-ID: comp-name, version-number
{ // Component Profile Specifications
Comp-Type: host-centered-comp | distributed-comp;
Platform-Property: <OS-Property>…;
Language-Property: <Lang-Property>,…;
[Conf-Property: <Conf-Property>,…<Conf-Property>;]
}

Component API specifications provide a component's API information, including function signatures, their input and output data types and parameters, return data types and parameters. In addition, some macro variables and values are included for C++/C-based components only.

Component-API: comp-name, version-number // API Specifications
{[<Include-Files>….;] // Class Files and Data Files
[Set-Macros: <Set-Macro>, … <Set-Macro>;] // for Macro Data Only
Provided-Functions: // Component API functions
<Function-Signature>; | <Condition-Function-Signature>;
……;}

The detailed syntax of <Function-Signature> is given below:

[return type] func-name (
[input-list: [data-type data-id, ….][class-type var-id,….]]
[output-list:[data-type data-id,…][class object-id,…]]
);

Component Interaction Specifications provide the details about the interaction interfaces to dependent components, including each port specification to its dependent components' functions. The syntax of a component interaction interface is given below.

Component-Interact-Interface: Comp-ID Version-No
{ <***Port-Spec***> , … ,<***Port-Spec***>; }

Port: Comp-Name, Version-No {
[<Include-Files>….;]
[Set-Macros:
<Set-Macro>, …, <Set-Macro>;] // for C++ or C only
Required-Functions:
<Function-Signature>,| <Condition-Function-Signature>,
….;}

The CIDL specifications for a component can be presented in different formats, such as BNF and XML. Based on a component's API specifications, the component CIDL manager (a function module in a component test bed) is used to parse the CIDL specifications, and generate a component test wrapper in the following steps: (The details can be found in [18])

- Parse the provide CIDL specifications to get the component and its API information.
- Create a component test wrapper source code file (named as XXWrapper.java for Java components) using a standard test interface template.
- Instantiate each function of a standard test interface by adding the generated source code based on its API signature.
- Insert the generated source codes into the created source code file for each test interface function to realize its detailed implementation.

Component Test Connectors

What is a component test connector? To perform unit tests for a non-independent component (NIC), engineers need to set up a standalone test environment. In component unit testing, test drivers and test stubs are needed. Clearly, using a standard component test interface and its supporting test wrapper is a good solution to reduce the complexity of creating component test drivers. However, the test harness costs for an NIC component depends on the complexity of its dependencies of other components. To deal

with this issue, we have introduced a concept, known as "component test connectors", to reduce the test harness in building component test stubs.

A ***connector component*** can be considered as a special type of non-independent components which can not be independently deployed and executed for application usage like domain-specific application components. However, similar to functional components, we must provide its users with a set of well-defined encapsulated functions to support its role as a bridge between two functional components. A typical example of connector components in a real world is a modem in a network system. A modem provides a bridge to link a specific network terminal device to a given network.

The ***purpose*** of using component test connectors is to set up a consistent interaction interfaces for a component test bed to support diverse NIC components to perform unit tests and component integrations. With well-defined component test connectors, engineers can use a systematic way to control, monitor, and simulate component interactions in its reuse contexts. A component test connector for a NIC component includes the following essential functions:

- It tracks and monitors its interactions with its dependent components interactions.
- It provides a consistent manageable and controllable interface to support the interactions between components in a standalone test environment.
- It provides a standard reusable connector interface to other components.

As shown in Figure 7, a component test connector consists of two parts: a) a reusable connector, which tracks and monitors its interactions with its dependent components, and b) a number of connecting entities to different components, which plays as adaptors to its dependent components. Each connecting entity (such as Connector-To-A) supports one interaction port in a component interaction interface. In our implemented component test platform, the component test connector is generated based on a component test library and the provided CIDL descriptions about a given component's interaction interfaces have given before.

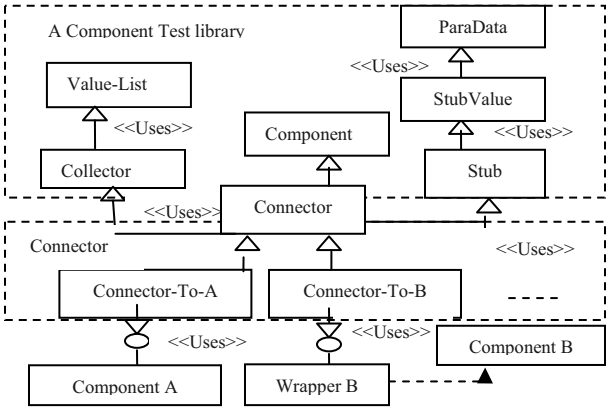


Figure 7 The relationship between components and connectors

As shown in Figure 8, component ElevatorController has a test connector, which connects to its two dependent components: Door and Floor-Panel. This connector consists of one common connector and one specific connector to each dependent component. In the proposed approach, the component test connector for each NIC component can be automatically generated based on the provided component's interaction interface descriptions in the CIDL. The detailed syntax for specifying a component's interactions is given before.

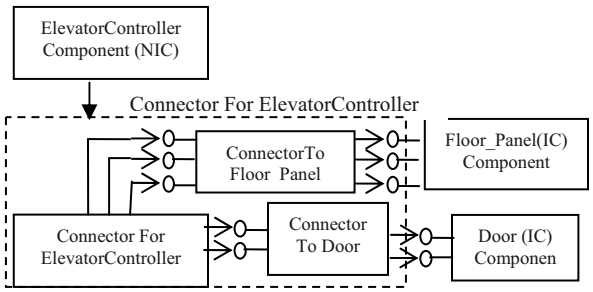


Figure 8 A Connector between ElevatorController and Others

The basic steps to generate a component test connector is listed below: (The details can be found in [18])

- Parse the provide CIDL specifications to get the component and its API information.
- Create a common test connector's source code file (named as XXConnector.java for Java components) using a standard test connector template.
- Create a specific test connector's source code file (Connector_To_XX.java) for each interaction port given in the CIDL.
- Instantiate a stub handler for each function of a given port based on its specified function signature in the CIDL adding the generated source code.
- Insert the generated stub handler into each specific connector.

C) A Component Test Framework

To support the standard component test interface and a component test bed, a component test framework is developed. Figure 9 shows the relationship among a testable component, component test framework, and other component test and management tool. This framework consists of five functional parts: a) component test suite, b) component test case manager, c) component test driver, d) component test controller, and e) component test result checker. It is made of a number of classes, which forms the basic component test library supporting the component test bed.

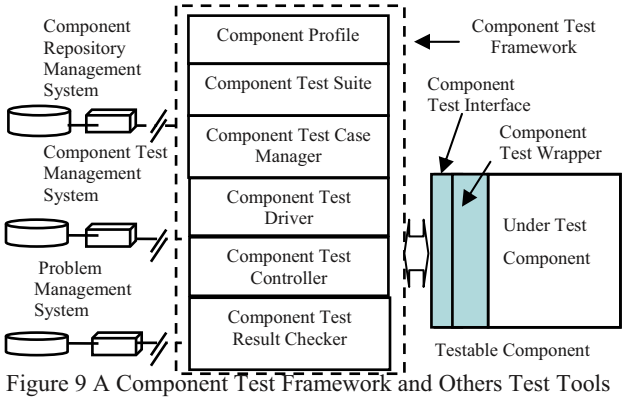


Figure 9 A Component Test Framework and Others Test Tools

D) A Component Test Tool for Component Test Automation

To support test automation of testable components, we need a reusable component bed to deal with diverse components with different APIs and technologies. Since 2005, we have developed a component test bed as a part of a component test tool (known COMPTEST) to support testable components. This prototype system has been redeveloped as a component unit test tool for

some production lines in **Huawei Technology Corp. LTD** to support unit testing of software components.

As shown in Figure 10, the generic component test bed for independent components (IC) includes the following parts:

- Component test library, which is made of the class library in the component test framework.
- Component test console, which is the user interface for component testers to support test operations.
- Component test adaptors, which is used to support different technologies and platforms.
- Component test executor, which supported with C-based scripts to exercise component APIs based on the selected components.
- Component test manager, which is a test directory that manages, stores, and maintains component unit tests (API-based test cases and test data).
- Component CIDL manager, which is a function module that manages, parses, and processes given standard component interface descriptions, written in the Component Interface Description Language (CIDL). Different formats can be used present component interface descriptions, such as BNF and XML.

Unlike other unit test tools, this component test platform has the following distinct features and advantages:

- Support different components with diverse APIs and standard component API descriptions in CIDL.
- Provide dynamic and reusable generated component drivers to exercise black-box component unit tests.
- Perform component unit tests in a plug-in-run-and-test manner.
- Component API-based black-box unit tests are not stored as a part of components, so component changes and test changes are independent.

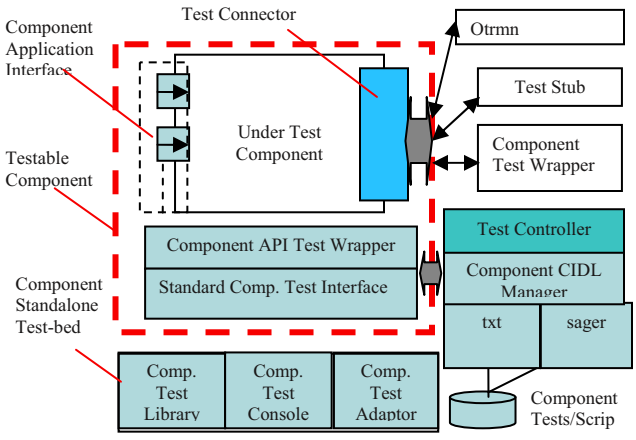


Figure 10 A Unit Test Platform (COMPTest) for Components

For a non-independent component, we need to set up a standalone unit test environment, which supports the component interactions with its dependent components. The component unit test platform in Figure 10 also supports non-independent components. For each under-test NIC component, a component test connector is created systematically to support the interactions with other dependent components. When a dependent component is not available, it can be simulated using a generated component test wrapper or manually generated component test stub. In the implemented component unit test platform, a controller is developed to control the component test connector to interact with other dependent components, stubs or their test wrappers.

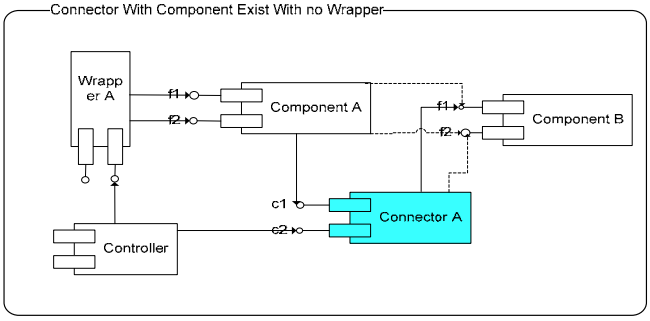


Figure 11 (a) Connecting a real dependent component B

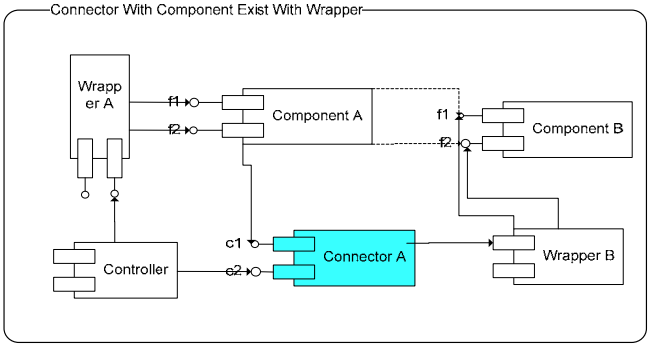


Figure 11 (b) Connecting with a dependent component's wrapper

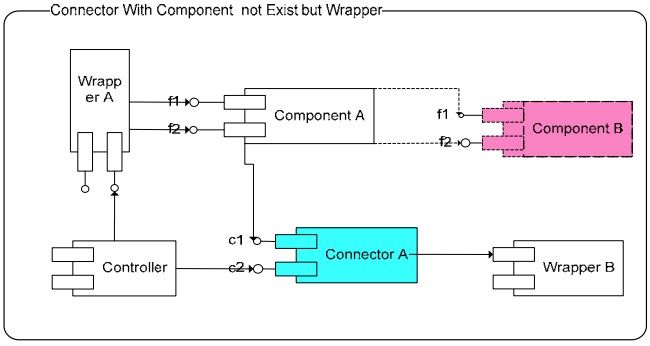


Figure 11 (c) Connecting with a component B's wrapper

In the second approach, we manually create component black-box API-based test drivers without the fix test data, as shown in Figure 12(a). The purpose of this approach is to develop the test drivers to accommodate similar component-based API-based tests based on component API function signatures only. Since test data are not coded inside test drivers, this approach reduces the dependency of the developed test drivers from specific test data. As shown in Table 4, 30 component API-based test drivers are developed in 280 minutes. Students spent about 34 minutes to execute the component tests. A standard component test connector not only interacts with dependent testable components, but also supports component's stubs, which simulate the functions of the dependent components. There are three applications of component test connectors. Figure 11 (a) shows the first scenario, in which a testable component A's connector interacts with its dependent component B directly to supports and track their interactions. The controller in the unit test platform controls the test connector for its

functions. Figure 11(b) shows the second application scenario, in which a component A's test connector directly interacts with the test wrapper of its dependent component B to track and support their interactions. Notice that component B is a testable component, its test wrapper can be generated based on component B's API descriptions in the CIDL. Figure 11(c) shows the third application scenario, in which the test connector of component A connects with the test wrapper of its dependent component B without B's existence. This scenario occurs when component B is not ready to be used for testing. Hence, its test wrapper can be generated and used to connect with some test tubs to support units testing of component A.

5. Application Experimental Results

To validate the proposed approach and developed component test platform, a group of mater students apply them onto a simple component-based elevator system, which is made of five components: a) an elevator, b) an elevator controller (ElevaContr), c) an operation panel (OpPanel), d) floor panels (FloorPanel), and e) Door component. Among of them, FloorPanel and door components are independent components. The elevator controller (ElevaContr) is a non-independent component, which is dependent on FloorPanel and Door components.

Table 2: A Statistic Report on Test Design for the Elevator System

Component Name	Function Name	BV Analysis Method	EQ Method	No. of test cases
FloorPanel	Floor_level	9	8	17
FloorPanel	Move	9	11	24
FloorPanel	Door_status	19	8	37
OpPanel	Elevator_level	9	8	17
OpPanel	Move	13	11	24
OpPanel	Stop	6	3	9
OpPanel	Move	6	3	9
OpPanel	Start	5	3	8
OpPanel	Dial	5	3	8
OpPanel	Door	10	5	15
ElevaContr	Floor_req	9	8	17
ElevaContr	Targetfloor_req	9	8	17
Elevator	Direction	10	8	18
Elevator	Move	9	8	17
Door	Door_status	19	18	37
Total	15	151	123	267

As shown in Figure 8, both Door component and Floor_Panel components are independent components. Component ElevatorController is a non-independent component with two dependent components (Floor_Panel and Door). Figure 8 shows the test connector of ElevatorController connects to its two dependent components through connecting ports. In our experimental case study, students use the two well-know black-box test methods (Boundary Value Analysis and Equivalence Partition) to design component API-based function test cases for the components of the Elevator system. Table 2 shows a statistic report about test case design for the five components of the Elevator System. There are 151 test cases driven using the boundary value analysis method, 123 test cases driven using the equivalent partition method.

To validate the effectiveness of testable components and COMPTest platform in reducing the costs of test harness. Students used three approaches to set up component test harnesses (test drivers and test stubs) to perform component API-based unit tests for the sample elevator system. As shown in Figure 12(b), the first approach uses the conventional approach to manually create component black-box API-based test drivers based on the test cases. All pre-defined test data are coded inside the component API-based function test drivers. As shown in Table 3, 30 component API-based test drivers are developed in 272 minutes. Students spent about 32 minutes to execute component tests.

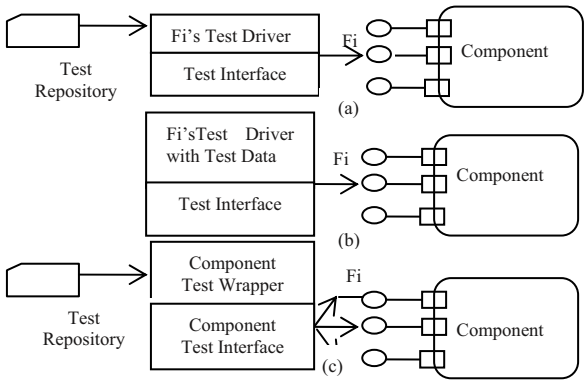


Figure 12 Three Ways to Access Component Tests

Table 3: Total Test Harness Cost Using Manually Generated Component Test Drivers with Test Data (in Mintues.)

Component Name	Function Name	Total No. of Created Test Drivers	Time to Run Tests	Develop Time for Test Drivers
FloorPanel	Floor_level	2	2	17
FloorPanel	Move	2	2	17
FloorPanel	Door_status	2	3	34
OpPanel	Elevator_level	2	2	17
OpPanel	Move	2	2	15
OpPanel	Stop	2	2	14
OpPanel	Move	2	2	14
OpPanel	Start	2	2	14
OpPanel	Dial	2	2	14
OpPanel	Door	2	2	17
ElevaContr	Floor_req	2	2	17
ElevaContr	Targetfloor_req	2	2	17
Elevator	Direction	2	2	15
Elevator	Move	2	2	16
Door	Door_status	2	3	34
Total	15	30	32 mins	272 mins

Table 4: Total Test Harness Cost Using Manually Generated Component Test Drivers without Test Data

Component Name	Function Name	Total No. of Test Drivers	Time to Run Tests	Develop Time of Test Drivers
FloorPanel	Floor_level	2	2	24
FloorPanel	Move	2	2	13
FloorPanel	Door_status	2	4	32
OpPanel	Elevator_level	2	2	25
OpPanel	Move	2	2	13
OpPanel	Stop	2	2	14
OpPanel	Move	2	2	14
OpPanel	Start	2	2	14
OpPanel	Dial	2	2	14
OpPanel	Door	2	2	27
ElevaContr	Floor_req	2	2	17
ElevaContr	Targetfloor_req	2	2	17
Elevator	Direction	2	2	13
Elevator	Move	2	2	11
Door	Door_status	2	3	32
Total	15	30	34 mins	280 mins

Figure 12(c) shows the last approach, which uses the proposed solution and COMPTest platform to execute the pre-defined

component API-based tests through a component test interface of testable components. Table 5 displays the different test harness costs. Students used the proposed solution to automatic generate 5 component API-based test drivers with the provided test data. They spent 73 minutes to develop and set up test drivers and 27 seconds to execute these component tests. As shown in Table 5, using the component testability tool (COMPTest) has a clear advantage over the other two conventional approaches in test driver development and test execution. It only needs 5 dynamically generated component test drivers with the minimum time (73 min.) in test driver development. In fact, the most time is spent on preparing API descriptions for all components using a given format.

Table 5: A Comparative View of the Case Study Results

Comparison Items vs. three approaches	Test driver with test data	Test driver w/o test data	Component testability tool
Total Number of Test Drivers	30	30	5
Max. Number of Source Code Lines	498	70	549
Min. Number of Source Code Lines	168	61	51
Total Number of Source Code Lines	4444	2986	5200
Total No. of Source Code Files	16	21	68
Total Development Time of Test Driver	286 (minutes)	272 minutes)	73 (minutes)
Component Test Execution Time	34 (minutes)	32 (minutes)	27 (seconds)

6. Conclusions and Future Work

This presents a systematic way to construct testable software components to increase component testability. The distinct contribution of this paper is its proposed component architecture, well-defined component test interfaces, as well as a systematic wrapping solution to convert COTS (or in-house-built) components into testable components. The proposed method has several distinct features:

- Convert a given COTS (or in-house-built) component into a testable component using a dynamically generated component wrapper with a well-defined component architecture. This component wrapper provides: (a) a well-defined component test interface to the external world, and b) a dynamic created internal test adapter to interact with the component APIs.
- Use a well-defined component test framework (which is a set of class library) as a middleware between the testable components and tools.
- Provide a reusable component test bed which interacts with testable components based on their common test interfaces.

Its major advantages are summarized below:

- Giving a practice-oriented solution to enhance component testability and a systematic approach to supporting component test automation.
- Offering a systematic way to construct a common test bed to deal with diverse COTS components.
- Providing a consistent component test interface between components and test tools.
- Reducing component test harness and costs to support its drivers in a component validation process.

Moreover, the paper reports our developed distributed component testing environment which supports test automation for testable components in component management, test management, test execution control, and model-based API test coverage monitoring and analysis. Unlike other existing test tools, our system has an intention to offer component users a plug-in-and-test solution to support component functional validation. In addition, the paper also reports our case study and application example of the proposed solution. The result indicates that this approach has a very good potential to reduce component test harness from users and allow them to validate COTS components using the same component test environment in a plug-in-and-test manner.

To carry this research into the next step, we are working on applying and extending this solution to other types of components, including graphic user interface components and communication-oriented components.

7. References

[1] Elaine J Weyuker, "Testing Component-Based Software: A Cautionary Tale", IEEE Software, September/October 1998.

[2] Sami Beydeda, Volker Gruhn , “Merging components and testing tools The Self-Testing COTS Components (STECC) Strategy “, Leipzig, Germany.

[3] Wang, Y. and G. King (2002), “A European COTS Architecture with Built-in Tests”, Proceedings of 8th International Conference on Object-Oriented Information Systems (OOIS'02), Montpellier, France, Sept., LNCS 2452, Springer, pp.336-347.

[4] Jerry Z. Gao, Jacob Tsao, and Ye Wu, Testing and Quality Assurance for Component-Based Software, Artech House Publishers, 2003.

[5] R. V. Binder, “Design for Testability in Object-Oriented Systems”, Communications of the ACM, September 1994.

[6] Roy S. Freedman, “Testability of Software Components”, IEEE Transactions on Software Engineering, Vol. 17, No. 6, June 1991.

[7] Jerry Z. Gao, et al, “Monitoring Software Components and Component-Based Software”, The proceedings of The Twenty-Fourth Annual International Computer Software & Applications Conference (COMPSAC2000), Taipei, Taiwan, October 2000.

[8] Jerry Z. Gao, K. Gupta, S. Gupta, and Simon Shim, “On Building Testable Software Components”, Proceedings of First Internation Conference on Cost-Based Software System (ICCBSS2002), pp.108-121, 2002, Orlando, FL, USA.

[9] Yingxu Wang, et al., “A Method for Built-in Tests in Component-based Software Maintenance”, The proceding of Third European Conference on Software Maintenance and Reengineering, IEEE Computer Society Press 1999.

[10] Stephen H. Edward, “A Framework for Practical, Automated Black-box Testing of Component-Based Software”, Journal of Software Testing, Verification and Reliability, Vol.11, No.2, 2001.

[11] Stephen H. Edwards, "Black-Box Testing Using Flowgraphs: An Experimental Assessment of Effectiveness and Automation Potential", December 2000 Issue of Software Testing, Verification and Reliability, Vol. 10, No. 4, pp. 249-262.

[12]Ram Chillarege, “Self-testing software probe system for failure detection and diagnosis”, The Proceedings of the conference of the Centre for Advanced Studies on Collaborative research, Toronto, Ontario, Canada, 1994.

[13] Yves Le Traon, et al., “Self-testable components: From pragmatic test to design-for-testability methodology”, The Proceedings of TOOLS (1999), Nancy, France, 1999.

[14] Jerry Gao and Ming-Chih Shih, “A Component Testability Test Model for Verification and Measurement”, The First International Workshop on Testing and Quality Assurance for Component-Based Systems (TQACBS05), 2005.

[15] R. Chhabra, R. Swamy, and W. Roberto Liang, “Case Study of a Systematic Solution for Testable Software Components”, Master Project Report, San Jose State University, May 2007.

SyncTest: A Tool to Synchronize Source Code, Model and Testing

Xiaoying Bai, Tao Liu
Department of Computer Science and Technology,
Tsinghua University, Beijing, China, 100084
baixy@tsinghua.edu.cn, tullyliu@gmail.com

Abstract

Model-based testing (MBT) is challenged by the synchronization between the source code, model and test scripts. When a change is made to the program, it is hard to identify the affected parts in the model and selectively re-run the affected test scripts. The research proposed a framework to enhance the traceability at two abstraction levels: the model level from program model to test model and the code level from SUT (Software under Test) source code to test scripts. The tool SyncTest was developed that can be used in the maintenance of a legacy C++ system. SyncTest derives the program model by program analysis and reverse engineering on the legacy system. It can automatically generate the U2TP (UML 2.0 Testing Profile) compatible test model in a rule-based approach and map the test model to test scripts. The test scripts written in C++ are associated to the target source code so that they can be compiled with and exercised on the SUT. The mapping information are tracked and managed for selective testing. Built on the Eclipse open source project, SyncTest provides an open platform that can be integrated and extended with other plug-in tools such as parsing, modeling and test generation tools.

Keyword: Model-based testing, Legacy system, C++ testing tool

1 Introduction

Testing is expensive, requiring a large amount of effort and resources. Automated testing is necessary to reduce the cost and increase productivity. Traditional test automation techniques are based on program analysis, such as control flow and data flow analysis. In recent years, the model-based testing (MBT) approach is evolving as a promising technique to address the challenges of automatic test cases generation [1][3][4][6][7][8][9][11] [12]. A model of the intended system behavior is established to serve as the basis for automatic test generation. In practice, the model can be abstraction at different levels including requirement model representing external observable software behavior, or design model representing internal software structure and communication. Various modeling techniques have been proposed for depicting software behavior, such as Extended Finite State Machine (EFSM), Specification Description Language (SDL), ESTELLE, and UML.

The research is motivated by the problems we encountered when applying MBT to testing large C++ legacy system which has been maintained and evolved for many years in a company. Due to the lack of formal specifications of the software structure, the system with growing size and complexity is very hard to understand, change and test. The project aims to provide a platform to facilitate automatic and systematic testing and evaluation for the maintenance of the legacy system.

An observation of the project is that an issue with the MBT approach is the synchronization between the code and the test scripts. In general, the model in MBT is manually developed and separated from the code which is expected to be easier to understand, verify and maintain compared with the SUT (Software under Test). There usually lacks of the tracking between the modeling elements and the source code. When software change, models and code are manually changed respectively and testing is expected to detect the inconsistent changes between the model and the code. However, it is difficult to find the set of test cases which can detect most change-related defects due to the lack of tracking information. Sometimes, the test engineers have to re-generate all of the test cases from the changed model and re-run the entire set of test cases on the changed SUT. The number of test cases is usually huge for large legacy systems. Sometimes, it is almost impossible for the re-generate-all and re-test-all approach.

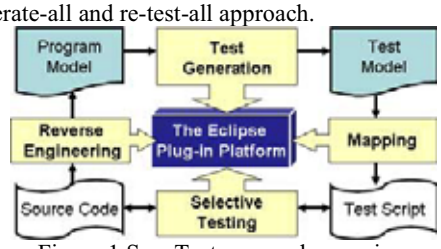


Figure 1 SyncTest approach overview

To address this challenge, the tool SyncTest is developed for C++ program testing to facilitate the MBT model construction, test generation, and selective testing. A framework is proposed to automatically capture the relationships among source code, model and test scripts, and track the mapping information for analysis. Figure 1 outlines the SyncTest approach which comprises the following four parts:

1. The OO program model is constructed by reverse engineering on the source code and program analysis techniques.

2. The test model is generated by transforming the structure and behavior specification of the OO program model to the test artifacts.
3. Test scripts are derived from the test model and coded in specific scripting languages such as C++ programming language.
4. Test scripts are associated with the source code so that they can be compiled with and exercised on the SUT. The associations are managed for related test case identification and selective testing.

In SyncTest, both of program model and test model are independent of platforms and programming languages. The models are specified following the XMI (XML Metadata Interchange) standard. The program models are constructed from the AST (Abstract Syntax Tree) representation generated by the program parser and are conform to the MOF (Meta Object Facility) meta-model of UML class diagram and sequence diagram. The test model specifies the test organization and execution concepts including test suite, test case, test objective, test data and test behavior, and are conform to the MOF meta-model of U2TP (UML 2.0 Testing Profile) [16].

SyncTest enables the synchronization of changes at two abstraction levels: the model level and the code level. At the model level, the translation mechanism is defined from the elements in the object oriented program model to the test concepts. At the code level, each test suite/test case is associated with an objective specification which traces to related target SUT file, version, class, and method definition. A rule-based approach is introduced to enable the flexible and extensible definition and interpretation of test strategies.

SyncTest is built as a plug-in project on the Eclipse open platform. It is integrated with a group of Eclipse [15] and SourceForge [17] open source tools including CDT (C/C++ Development Tooling) for source code parsing, program compilation and building; EMF (Eclipse Modeling Framework) for modeling; and CppUnit [13] for harnessing and running the test scripts.

The rest of the paper is organized as follows. Section 2 introduces the SyncTest test framework. Section 3 presents the implementation of the testing tool. And section 4 concludes the paper.

2 The SyncTest Test Framework

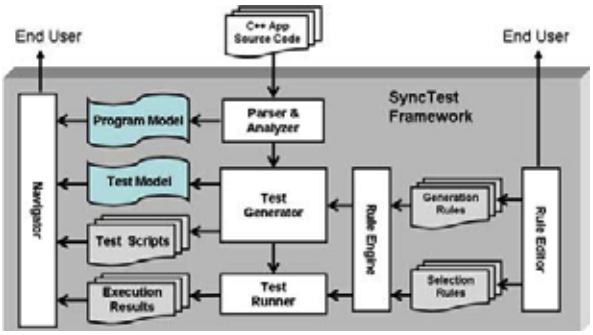


Figure 2 SyncTest framework overview

As shown in Figure 2, the SyncTest Framework is composed of following three layers: model construction by reverse engineering, model-based test generation, and selective testing.

2.1 Model Construction by Reverse Engineering

It is the process of parsing and analyzing the C++ application source code, extracting the information and relationship of software components, and constructing model specifying software structure and behavior, for better understanding and maintaining the software system. To achieve the goal, following problems have to be solved: (1) how to gather the necessary information; (2) how to model and specify the model of the gathered information; and (3) how to visualize the model.

SyncTest extracts the information of classes, methods, dependencies and interactions based on the AST representation generated by the C++ parser Eclipse CDT [15]. The models can be (1) software static structures such as class definition, class hierarchy, source file dependency, and so on; and (2) software dynamic behavior such as message passing and interactions. As a de facto industry standard, UML is a good choice for model representation. An objective of UML 2.0 is to support "an approach to developing software that shifts the focus of development from code to models and to automatically maintaining the relationship between the two". SyncTest translates the structure and behavior information to the corresponding UML class diagram and sequence diagram conforming to the MOF meta-model encoded in the XMI format [16].

2.2 Rule-Based Testing

Model-based test generation is the process to generate test data, test procedures and test cases. SyncTest incorporates the rule-based mechanism to support definition and interpretation of the test generation strategies. Different categories of rules can be defined including the methods for generating test data and test procedures, the mapping from source code to test model, the naming conventions for generated test files, and the criteria for selective test run. Figure 3 shows an example XML specification of the rule for naming convention definition.

```
<rule>
        <testStrategy><name>SourcecodeAnlysis</name></testStrategy>
        <scope><name>ALL</name></scope>
        <testclass>
         <prefix>ts_</prefix><suffix>TestSuite</suffix>
                <mapping><left>TestClass</left></mapping>
        </testclass>
        <testmethod>
                <prefix>tc_</prefix><suffix>TestCase</suffix>
                <mapping><left>TestMethod</left></mapping>
        </testmethod>
        <testfile><prefix>test_</prefix><suffix></suffix></testfile>
        <folderName>test</folderName>
</rule>
```

Figure 3 Example rule specification

The externalization of rule definition and interpretation enables flexible and dynamic editing and binding to the rules through the rule editor and rule engine.

2.3 The Test Model

Test model is necessary to provide a standard definition and description of test information such as the organization structure of test suites and test cases, test data definition, test plan and scheduling, etc. It is specified independent of

the platform and programming language of the SUT, and thus can be easy to be maintained and reused.

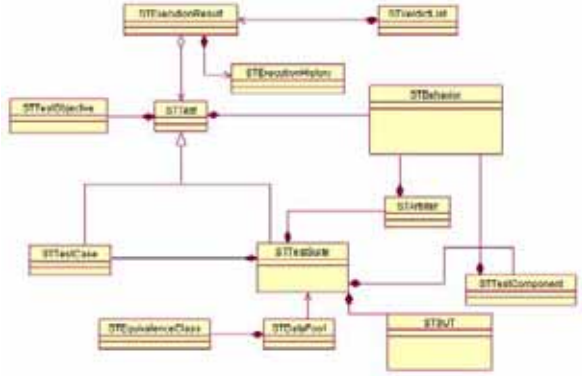


Figure 4 SyncTest test model

SyncTest supplies a platform independent test model and applies it for C++ test generation. As shown in Figure 4, the model, conforming to MOF-based U2TP metamodel, composes of following concepts:

- *STTestCase* is a definition of tests on the SUT with conditions, inputs, expected outputs, and a set of behaviors;
- *STTestSuite* is a group of test cases and it can control the order of running the test cases such as sequence, iteration, parallel, etc.;
- *STTestComponent* is a realization of a test case / test suite for executing it behavior;
- *STTestObjective* defines the target of a test case;
- *STBehavior* describes the execution of a tests;
- *STArbiter* provides the evaluation of test results; and
- *STSUT* represents the interface to the SUT.

The model is encoded in XMI specification as shown in Figure 5.

Figure 5 Example test model XMI specification

2.4 Selective Testing

For a large legacy system, the number of test cases is usually huge. When a feature is changed and regression testing is required, it is expensive to rerun all the test cases. Sometimes, it is even impossible. Selective testing is necessary to reduce the cost by allowing the user to select and run a small subset of test cases according to certain criteria. It is also helpful for analyzing the failures and locating the defects with a smaller set of test results compared with the run-all approach.

SyncTest tracks the dependencies between test elements and SUT through the test model. At the model level tracking, different generation and mapping rules may be defined such as:

- To generate a test suite for each class and a test case for each method in the class;
- To generate a test suite with multiple test cases for each method; and
- To generate a group of test suites for each class.

At the code level, with the help of STTestObjective, SyncTest provides the tracing information between the test suite/test case and its target SUT files, file versions, classes, and methods.

3 SyncTest Implementation

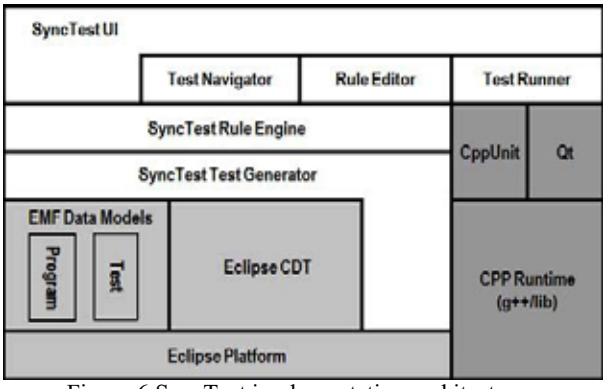


Figure 6 SyncTest implementation architecture

As shown in Figure 6, SyncTest is implemented as a plug-in project on the Eclipse open source platform. It is integrated with CDT and EMF plug-ins [15]. CDT plug-in provides the IDE for editing, compiling and executing C++ applications. EMF plug-in supports the modeling and representation of SyncTest program model and test model. SyncTest also integrates with CppUnit unit test framework for generating the test scripts and CppUnit Qt TestRunner for executing the test scripts. Figure 7 is the screen of SyncTest test generation, edit, compiling, execution, navigation and management.

3.1 Interfacing to CDT

The syntax of C++ is much more complex than other object oriented languages such as Java. For examples, the macro mechanism in C++ requires additional pre-processor, and the flexible combination of pointers and templates may need special syntax analyzer. The capability of the parser is critical for code analysis, information extraction and UML model construction. CDT is an Eclipse open source project and an industrial strength C/C++ IDE. It provides the parser, known as the DOM (Document Object Model) parser, to analyze the C++ source code and generate the AST as CDT's internal representation of the code. The parser can

process most of the GCC extension grammar, and preprocess the include files, the macro definition and expansion. CDT also provides the IDE environment for search, code navigation, compilation, and build assist.

SyncTest integrates CDT from two aspects: the front end AST generator and the build system to compile and link the test scripts coded in C++ programs with the SUT to generate executable applications. CDT parser does not perform any semantic analysis or type-checking during the parse. SyncTest translates the specifier declarations (such as IASTClassSpecifier, IASTFunction, IASTMethod, IASTTemplate, etc) recognized in the AST to the SyncTest object oriented modeling elements. SyncTest also uses the information of parser pre-processing, such as file dependency analysis, to generate the include declarations that bind the test scripts to the SUT.

3.2 Interfacing to EMF

SyncTest specifies and maintains the program models and test models using the EMF mechanisms. EMF provides a modeling framework and the code generation facilities. It can facilitate the specification of the data model, the transformation of the models into Java code, and the serialization of the model to persistent data. It also automatically maintains the dependencies between modeling elements. Each element keeps a list of its observers which can be notified whenever a state change occurs.

3.3 Interfacing to CppUnit

```
// cppunit_cpp.vm
\#include "$filename"
CPPUNIT_TEST_SUITE_NAMED_REGISTRATION($classname,"$classname")
void $classname::setUp(){}
void $classname::tearDown(){}
#if($methods)
#foreach ( $method in $methods)
${method.Ret}
$classname::${method.Name}(${method.Params}){}
#end
#end
```

```
// cppunit_h.vm
#ifndef ${filename.toUpperCase()}_H
#define ${filename.toUpperCase()}_H
\#include <cppunit/extensions/HelperMacros.h>
class ${classname}:public CppUnit::TestFixture{
CPPUNIT_TEST_SUITE( ${classname}TestSuite );
#if($methods)
#foreach ( $method in $methods)
        CPPUNIT_TEST( ${method.Name} );
#end
#end
        CPPUNIT_TEST_SUITE_END();
public:
        void setUp();
        void tearDown();
#if($methods)
#foreach ( $method in $methods)
${method.Ret} ${method.Name}(${method.Params});
#end
#end
};
#endif // ${filename.toUpperCase()}_H
```

Figure 7 Test script template definitions

SyncTest uses the CppUnit [13] test framework as the template for the generated test scripts. CppUnit is the C++ port of the JUnit framework for unit testing. It is motivated by the Test-Driven Development (TDD) principle, which emphasizes daily, even hourly, continuous build and testing. In the framework, test cases are bound to the classes and methods to be tested. The programmers develop test together, or even earlier than, the program code. It can help change impacts analysis during regression testing to identify the affected the code and test cases.

SyncTest defines the scripting template for test suites and test cases based on the test facilities in CppUnit. It uses the Velocity template engine of Apache project to generate instances of test suites and test cases according to the scripting template. Figure 7 shows the .h and .cpp file of the template definition.

SyncTest also integrates with CppUnit Qt TestRunner for selective executing the test cases. The CDT compiler accepts the source code and test files, incorporates the DLL of CppUnit and Qt TestRunner, compiles all together and generates the executable test program. SyncTest also extends the TestRunner to output and save the execution results in XML-encoded reports.

3.4 Test Navigation and Management

SyncTest provides a unified navigator for browsing the source code, the test model and test scripts. Internally, the test model traces the mapping information between the source code, the model and the tests. With the build-in dependency management mechanism supported by EMF, SyncTest can support an effective way to organize and manage the tests hierarchically with multiple views. Users can brows, query, and locate all the related information conveniently.

4 Experiments

Figure 8 illustrates the process from the source code files to identified the class model, to the generated test model, and finally to the coded test scripts in C++ programs.

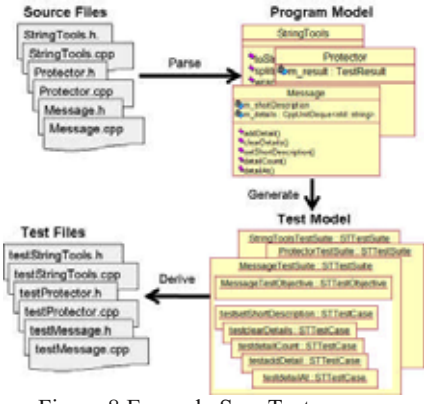


Figure 8 Example SyncTest process

Experiments are exercised on many C++ projects. Table 1 shows the experiment results (including the number of identified classes and methods, and that of generated test suites and test cases) on the following four SourceForge open source projects:

- CppUnit is a project of C++ unit testing framework with 72 header files and 53 cpp files;
- Bouml is a UML modeling tool with 793 header files and 766 cpp files;

- FileZilla client is a FTP client tool with 110 header files and 113 cpp files; and
- eMule is a P2P resource sharing tool with 308 header files and 246 cpp files.

Table 1 Experiments summary

SUT		Cpp-Unit	Bouml	FileZilla Client	eMule
Source File	Header	72	793	110	308
	CPP	53	766	113	246
Program Model	Class	57	836	182	353
	Method	429	4878	2113	6126
Test Model	Test Suite	57	836	182	353
	TestCase	361	4729	1967	5771
Test File	Header	57	836	182	353
	CPP	57	836	182	353

The strategy in the experiment is to generate a test suite for each class and generate a test case for each method. The result showed that the tool correctly indentified all the classes, class inheritance relationships, methods and methods signature for structure analysis. Test suites and test cases are generated to cover the structure features at the class interface level.

5 Related Works

5.1 Reverse Engineering

In general, different approaches exist for information gathering including program analysis such as data flow and control flow analysis, compiler technique such as AST-based analysis, and source code instrumentation.

Many methods were proposed to reverse engineer the static and dynamic information and represent the information using UML diagrams such as class diagram and sequence diagram [1][6][7][9][10][12]. For example, L.C. Briand et al. [1] reported the research on reverse engineering of UML sequence diagrams by instrumenting the source code to capture the messages including method entry and exit, conditions, and loops. On running a pre-defined scenario, the instrumented code produced the tracing logs, which are further mapped to the UML sequence diagram based on the mapping rules defined by OCL (Object Constraint Language). A. Rountev et al. [10] proposed an extension to UML 2 control flow primitives and discussed the algorithm to precisely map inter-procedural flow of control to UML sequence diagram using the proposed extensions. B. A. Malloy and J. F. Power [7] introduced the tool SPIDOR for analyzing and modeling C++ program dynamic behavior with UML sequence diagrams. SPIDOR used GNU Compiler Collection, GCC, as its front end and took GCC AST GENERIC for class hierarchy and call graph analysis. It also used aspects to insert probes into the code for profiling the interactions of objects and methods.

SyncTest aims to integrate the traditional reverse engineering techniques with MBT and to improve the maintainability of legacy systems.

5.2 Rule-Based Testing

The research of applying the rule-based approach to testing can be traced back to Deason et al.'s work on automatic test data generation in the early 1990s [2][5]. In the proposed paradigm, the rule interpreter took the rules defined in the rule base with the symbol table gathered by the program parser, and generated the test cases to meet the test adequacy criteria including condition coverage, decision coverage, and multiple-condition coverage. The experiment results showed that with properly defined rules, it can significantly reduce test cost while improving test coverage compared with the random and statistical testing approaches.

SyncTest takes the rule mechanism as part of the infrastructure to guide the generation and execution of test cases. Rules can be defined at different levels and granularities, and applied at different stages of the SyncTest approach including the transforming from program model to test model, the generation of test scripts, and the selective run.

5.3 Test Modeling

U2TP [16] provides a standard definition of the basic testing concepts. *Test Architecture* which specifies the structure of a test context covering test components, SUT, test configuration and scheduler. *Test Behavior* which specifies the objectives, invocation, execution, oracle, control, and coordination for exercising a test. *Test Data* which specifies the data used in stimuli to and observations from the SUT, and for coordination between test component, including data pool, data partition, data selector, and coding rule. *Time* specifies the time constraints, time observations and/or timer within test behavior specification. U2TP defines the test modeling language using the UML meta-modeling approach. A MOF meta-model is defined enabling the use of U2TP independent of UML.

Eclipse™ Test and Performance Tools Platform's (TPTP) testing tool provide a reference implementation of the U2TP based on EMF. The TPTP test model composes of the core models of testing profile, behavior and execution. However, the current TPTP test model is tightly bound to Java implementation and integrated with JUnit testing framework. To reuse the test model in the open test platform, it is necessary to decouple it from Java binding and extend it as a language independent model.

SyncTest defines a test model that is conform to U2TP MOF meta-model, and provides an implementation independent of programming languages using EMF.

6 Conclusion and Future Work

Due to the complexity of C++ language, it is hard to understand, change and test a legacy C++ system. Tools are very few for C++ program maintenance and automatic testing compared with OO languages. The paper introduced the SyncTest tool which is built as a plug-in project on Eclipse open platform to streamline and automate the process of source code analysis, program model construction, test model generation and test script generation. It tracks the transformation process from program model to test model, and the correlations between code and test scripts. In this way, it enables change impact analysis at the model level as well as the code level and has the potential to enhance the efficiency of regression testing form legacy system maintenance.

The paper reported our first attempt of an on-going project on the SyncTest framework and tool development. The running prototype and some early experiment results show the promise and potential benefits of the proposed approach. However, a lot of works remain from both research and practice perspectives including (1) behavior modeling and model transformation mechanism from OO program model to test model; (2) sophisticated rule definition and rule-based test generation algorithms; and (3) experiments and measurements for effectiveness comparison and evaluation.

7 Acknowledgement

This research is sponsored by Freescale Semiconductor Inc. We are grateful to Dr. Kwok Wu, Dr. Avinash Naidu, and Dr. Xinxin Yang for their discussion of the industry problems and constructive comments on the research and implementation.

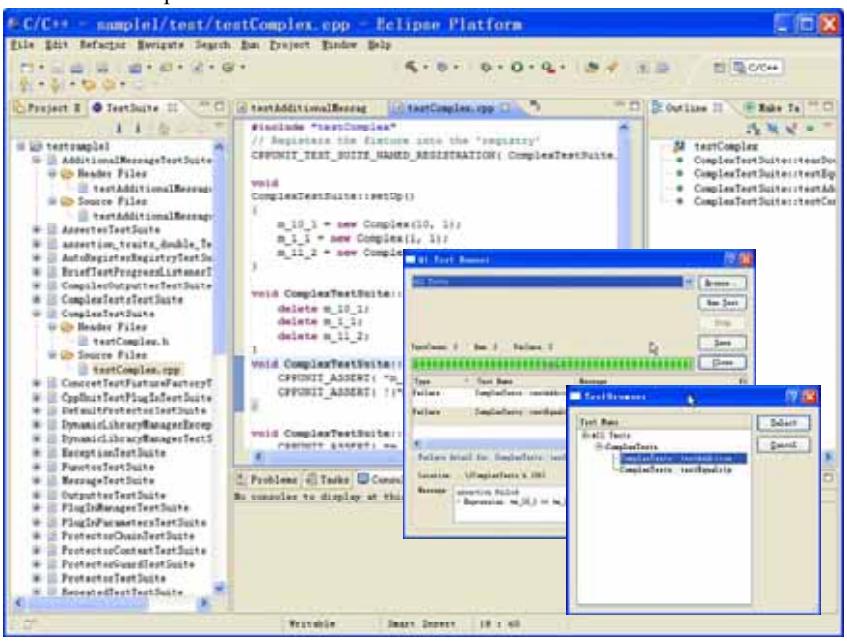


Figure 6 The screenshot of SyncTest

References:

[1]. L. C. Briand , Y. Labiche ,and Y. Miao, "Towards the reverse engineering of UML sequence diagrams", *Proceedings of the 10th Working Conference on Reverse Engineering*, 2003, pp.57-66.

[2]. K. H. Chang, W. H. Carlisle, J. H. Cross, and D. B. Brown, "A heuristic approach for test case generation", *Proceedings of the 19th ACM Annual Conference on Computer Science*, 1991, pp. 174-180.

[3]. I. Craggs and C. Beverland, "Model-based testing for object-oriented programming interfaces", *Proceedings of the Workshop on Model-based Testing and Object-Oriented Systems*, Technical Report MSR-TR-2006-148, Microsoft Research, 2006, pp. 4-11.

[4]. S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model-based testing in practice", *Proceedings of the International Conference on Software Engineering*, 1999, pp. 285-294.

[5]. W.H. Deason, D.B. Brown, K.-H. Chang, and J.H. Cross, "A rule-based software test data generator", *IEEE Transactions on Knowledge and Data Engineering*, Vol 3, No. 1, 1991, pp. 108-117.

[6]. F. Fraikin and T. Leonhardt, "SeDiTeC-testing based on sequence diagrams", *Proceedings of the 17th IEEE International Conference on Automated Software Engineering*, 2002, pp.261-266.

[7]. B. A. Malloy and J. F. Power, "Exploiting UML dynamic object modeling for the visualization of C++ programs", *Proceedings of the 2005 ACM symposium on Software visualization*, 2005, pp. 105-114.

[8]. A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner, "One evaluation of model-based testing and its automation", *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 392-401.

[9]. M. Riebisch, I. Philippow, and M. G, "UML-based statistical test case generation", Net.Object.Days 2002, LNCS Vol. 2591, pp 394-411.

[10]. A. Rountev, O. Volgin, and M. Reddoch, "Static control-flow analysis for reverse engineering of UML sequence diagrams", *The 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE 2005)*, 2005, pp. 96-102.

[11]. M. Utting, A. Pretschner, and B. Legeard, *A taxonomy of model-based testing*, Technical report, Department of Computer Science, University of Waikato, New Zealand, 2006.

[12]. Y. Wu, M. H. Chen, and J. Offutt, "UML-based integration testing for component-based software", *Proceedings of Second International Conference on COTS-Based Software Systems*, 2003, pp. 251-260.

[13]. CppUnit, 2008, http://cppunit.sourceforge.net/.

[14]. Apache Velocity Project, 2007, http://velocity.apache.org/.

[15]. Eclipse platform, 2008, http://www.eclipse.org/.

[16]. OMG standard, 2005, http://www.omg.org/.

[17]. Sourceforge, 2008, http://www.sourceforge.net/.

A Virtual Machine for Distributed Agent-Oriented Programming

Bin Zhou
School of Computer, National Univ. of Def. Tech.
Changsha, China, Email: binzhou@nudt.edu.cn

Hong Zhu
School of Technology, Oxford Brookes University
Oxford OX33 1HX, UK, Email: hzhu@brookes.ac.uk

Abstract. *Agent-orientation has been considered as a viable solution to the development of software systems in dynamic environments such as the Internet. This paper presents a high level language virtual machine CAVM designed for distributed agent-oriented programming in the Internet environment. The main features of the virtual machine (VM) are two-fold. First, the communication between agents is separated from computation so that communication is network transparent of agent location. Second, code deployment is separated from loading so that multiple agents of the same caste can be dynamically distributed to the network and dynamical integrated into the systems by adding new agents. The paper first reviews the key features of an agent-oriented programming language called CAOPLE. It then presents the design of the virtual machine to support the implementation of the language. Experiments with the performance of the system in a network environment are also reported.*

1. Introduction

Agent-orientation has been long considered as a viable solution to the development of software systems in dynamic environments such as the Internet [1, 2]. A great amount of research efforts has been reported in the literature; see, for example, [3,4,5]. However, the IT industry has been slow to adopt the approach. A key problem that hampers the wide adoption of agent technology is the lack of efficiently implemented agent-oriented programming languages. Among the most promising approaches to the design and implementation of such a language is virtualization, which can provide a high level abstraction of computation resources associated to the internet and a unified framework for efficient uses of the resources [6]. The most appealing feature of virtualization is that it can provide software developers and end-users a virtual computation environment that is conceptually simple and easy to use through hiding the complexity caused by the heterogeneity and spatial distribution of hardware and the diversity of software platforms and interaction protocols.

In this paper, we present the design and implementation of a virtual machine called CAVM, which stands for Caste-centric Agent-oriented Virtual Machine. It is a high level language virtual machine (VM) for the implementation of a high level agent-oriented programming language, CAOPLE, for distributed programming on the Internet. The caste centric model is a simple but powerful multi-agent model of software systems proposed in [7]. Its expressiveness has been demonstrated by the specification and modeling of various types of multi-agent systems, communication protocols, distributed algorithms and web services architecture and applications [7, 8].

The remainder of the paper is organized as follows. Section 2 briefly reviews the caste-centric model on which the VM is based, where caste is the modular program unit and the templates of agents. The key features of an agent-oriented programming language CAOPLE will be described. Section 3 presents the design of CAVM, the VM in order to efficiently implement the semantic of CAOPLE. Section 4 describes the implementation of the system and reports the main results of preliminary evaluation. Section 5 concludes the paper with a discussion of related work and future work.

2. Overview of The CAOPLE Language

This section briefly reviews the key concepts and features of the caste centric model [9, 10] and the language CAOPLE and discusses their requirements on the VM.

2.1. The Caste-Centric Model of Multi-Agent Systems

In this model, a software system consists of a number of active autonomous computation entities called *agents*. Agents are instances of Castes, and may be distributed over a network and execute concurrently. Each agent encapsulates four parts, which are defined in their corresponding Caste:

- *State* space defined by a set of variables;
- *Actions* that the agent can perform;
- *Behaviour rules* that the agent uses to determine when to take an action and how to change its state; and
- *Environment description* that defines the entities in the system the agent observes.

An action can be either *observable* by other agents or just *internal*. When an agent takes an externally observable action, it generates an event that the outside can perceive. Similarly, a state variable can also be either *observable* by the outside or just *internal*. The outside can obtain (but not change) the value of an observable variable. Thus, agents communicate and interact with each other through taking observable actions and changing observable states and observing other agents' actions and states. It is worth noting that in this model, all entities in a multi-agent system are agents. Object can be considered as a degenerate form of agent [7, 9].

For example, the following simple CAOPLE program given in Figure 1 defines a caste `GreetingAgent`, whose agents are capable of taking two actions, to say `Hello World` and to say `Welcome`. Each of them observes all other agents of the caste. Their behaviour rules are: (1) to say `HelloWorld` when it is created, and (2) whenever it observes another agent says `HelloWorld`, it responds with `Welcome`.

```
caste GreetingAgent;
  observes all GreetingAgent;
  action HelloWorld; Welcome;
  init HelloWorld;
  body
    when some A in GreetingAgent: HelloWorld()
      -> Welcome() End;
end GreetingAgent
```

Figure 1. The HelloWorld program in CAOPLE

As shown in the HelloWorld example, agents are defined as instances of castes. A caste defines a template of agents via encapsulating a collection of structural and behavioural characteristics in the form of a set of state variables, a set of actions, a set of behaviour rules and a description of a set of other agents as its designated environment. Here, caste plays a similar role as class in Object-Oriented (OO) programming and data type in structured programming. However, in OO paradigm an object is bounded to its class statically. In contrast, in the caste model, an agent is bounded to its castes dynamically, i.e. it may change its caste membership at runtime by joining to or quitting from a caste. An agent can also be a member of a number of castes at the same time. The CAOPLE program given in Figure 2 defines a caste Creator whose instances will create a number of agents of caste GreetingAgent to populate the world.

The location of the caste in a create statement can be specified explicitly or implicitly. The general format of a create statement is

create [AgentName in] CasteName(Para) [@URL],

where URL is a string that gives the location where the caste object code is deployed. When URL is omitted, the location of the caste must be resolved during the deployment of the caste through a search strategy. However, agents of the same caste can be created and execute on different machines. Thus, the executable code of a caste must be loaded to these machines from where it is deployed. Such distributions of code may happen at runtime. Agents can be created at runtime and joins a caste at runtime through remote loading of the object code from where the caste is deployed.

For example, suppose that two agents of caste Creator declared in Figure 2 run on machine C_1 and C_2 and generate N_1 and N_2 agents of GreetingAgent, respectively. The system will then contain a total of N_1+N_2 agents of GreetingAgent; N_1 agents on C_1 and N_2 agents on C_2. They must be able to communicate with each other despite of their distribution on the different computers. Moreover, the system must also support the integration of new agent into the system when a new agent is created. For example, if another new agent of GreetingAgent is created on a computer, say C_3, all the N_1+N_2 existing agents should respond to its HelloWorld action with their Welcome actions. This simple example shows that CAOPLE has the features of network transparency of agents' location because the programmer does not need to know where the agents are located at runtime.

The caste centric model not only supports dynamic integration of new agents into a system, but also adaptation of behaviours through dynamic casteship changes. For example, suppose that three sub-castes of caste GreetingAgent are defined as in Figure 3. An agent of caste Smart can determine its behaviour according to the weather of the day. When the weather is fine, it will join the caste GolfPlayer and invite the new agent to play golf. If the day is rainy, it will join the caste of CoffeeDrinker and invite the new agent to drink coffee.

The support to the seamlessly dynamic integration of new agents into the system must also enable new castes to be added into the system without interfering with the existing

```
caste Creator (population: Integer);
  init
     Begin for i:=1 to population do
        create GreetingAgent;  end;
end Creator
```

Figure 2. An example of dynamic creation of agents.

```
caste GolfPlayer is GreetingAgent;
  action InvitePlayGolf();
  body
    when some A in GreetingAgent: HelloWorld()
      -> InvitePlayGolf();
    End;
end GolfPlayer
caste CoffeeDrinker is GreetingAgent;
  action InviteCoffee();
  body
    when some A in GreetingAgent: HelloWorld()
      -> InviteCoffee();
    End;
end CoffeeDrinker
caste Smart is GreetingAgent;
  observes WeatherMan in WeatherForecaster
  body
    when some A in GreetingAgent: HelloWorld()
      -> if WeatherMan.Today='Fine'
         then join(GolfPlayer)
         elseif weatherman.Today='Rainy'
         then join(CoffeeDrinker)end;
    End;
end Smart
```

Figure 3. An example of adaptive behaviour

ones. For example, the caste Monitor given in Figure 4 can be written and compiled after the agents of castes GreetingAgent and Creator are created and running. When an instance of the Monitor is created on a machine in the network, it will start to count how many Welcome actions are taken by the agents of caste GreetingAgent no matter whether the agents are created before or after its creation.

The overall structure of CAOPLE programs consist of a number of type declarations and caste declarations.

A type declaration defines the formats of data that are exchanged among agents, such as the parameters of observable actions and the values of observable variables. The data are represented in XML. The type definition defines the formats in Pascal-like syntax. It takes both advantages of the flexibility and extendibility of data representation in XML and the readability and high level of abstraction of type definitions in Pascal-like programming languages and enables static type checking during compilation. A type definition can be easily translated into XML for runtime processing. Details are omitted here as the focus of the paper is on the VM.

2.2. Requirements on the Virtual Machine

In order to support distributed programming in a network environment, the VM must support the following key features of the CAOPLE language facilities.

Distributed deployment. Object code of a caste must be deployed to a unique location in a distributed computer system so that consistency of the code can be managed. Object codes of different castes must be able to be deployed to different computers so that load balance can be achieved. Dynamic deployment must also be supported so that new caste can be deployed without interfering with the execution of an existing system.

```
caste Monitor;
  observes all GreetingAgent;
  var counter: Integer;
  init counter :=0;
  body
     when some A in GreetingAgent:
             SayWelcomeToTheWorld()
       -> counter:=counter+1 end;
end Monitor
```

Figure 4. An example of dynamic integration of castes

Dynamic remote loading. An agent must be able to be created or join a caste dynamically through create/join statements. Consequently, the deployed caste object code must be able to be loaded to any computer in the system at runtime. When multiple agents of the same caste exist on the same machine the code will be shared by these agents rather than storing duplicated copies.

Autonomic management of object code. An agent can be destroyed or quit from a caste using destroy/quit statements. The loaded object code may be no longer needed thus can be removed from the machine. However, the object code could still be required as other agents may remain alive and running on the same machine. Such management of loaded object code must be performed autonomically.

Transparent communication. In CAOPLE, agents communicate with each other through taking observable actions and observing other agents' states and actions in their environments. This communication facility is highly abstract and transparent to the location where the agents are located. This mechanism is essentially event driven. An agent's observable actions can be considered as publication of events. The environment description can also be understood as subscription to such publications. This publication/subscription mechanism must also be supported by the VM.

3. The Virtual Machine CAVM

This section presents the design of the virtual machine.

3.1. Architecture

As illustrated in Figure 5, CAVM consists of two types of components: *Local Execution Engines* (LEEs) and *Communication Engines* (CEs). The LEEs support the executions of agents while the CEs support the communications between agents, which may share the same computer with an LEE (e.g. CE_1 and LEE_1 in Figure 5) or on different computers over a network (e.g. CE_2 and LEE_2).

A program written in CAOPLE that consists of a number of castes is compiled into CAVM's object codes. Each caste's object code is deployed to one CE, but can be loaded to a number of LEEs at runtime. When an agent of the caste is created or an agent joins the caste on an LEE, the object code is loaded if it is not already there. The object code could be loaded locally or from a remote CE.

An object code file generated by compiler contains the definition of a single caste in the object code of the CAVM. It includes three main sections: constants, initialization code and body code. The constant data section contains literal constants and reference addresses in the code sections, such as the offsets of state variables, offsets of action bodies, offsets of environment variables, etc. The initialization code section contains the instructions for the initialization of agent when the agent is created or joins the caste. The body code section contains the instructions fulfilling the main functionalities of the agent. It is compiled from the source code in the Body part of the caste. The object codes are represented in XML format, which is transformed into a binary format when the code is loaded to LEE.

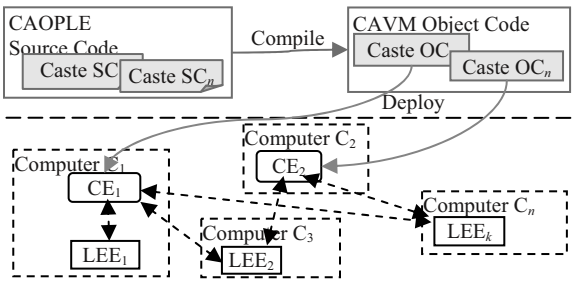


Figure 5. The architecture of CAVM

Caste deployment is mandatory before any agent can be instantiated from it. It binds the object code of a caste to a communication engine CE. The process consists of two steps. First, the CE stores and registers the caste's object code file and second the CE sets up and initializes the membership management service and the communication services for the caste.

If a caste is deployed successfully on a CE, we say that the CE is the host CE of that caste. In general, a CE instance can host many resident castes.

3.2. Local Execution Engines

As shown in Figure 6, a local execution engine (LEE) consists of the following components. Program space (PS) stores the object code of castes loaded on the LEE together with LLC, a list of stored castes and their locations in the program space. Loader finds and loads the object code of castes into the program space when instructed by the Central Processing Unit (CPU). A pre-defined search policy is applied by the Loader to locate the object code deployed on CE. Memory Space (MS) is the runtime memory that stores the states, environment data of the agents running on the LEE, organized as agent context data (current program counter, operand stack and local variables, etc). When an agent quits from a caste, its context data is discarded. CPU interprets instructions stored in the PS and processes the data stored in the memory space. For each instruction, the CPU changes the state of the memory space and context register and updates the Program Counter (PC) and then loads the next instruction to the CPU. PC is a pointer to a location in the PS where the next instruction will be loaded to the CPU to execute. It therefore represents a thread of control. Upon sending/receiving state/action update messages to/from a particular CE, environment data is updated autonomically and asynchronously by the Communication Manager.

CAVM supports not only parallel computation by running a number of LEEs and CEs on a network of computers, but also concurrent execution of multiple agents on one computer through interleave. The multiple threads of executions are achieved through a schedule policy (currently, round robin) and switches between agents using the Context

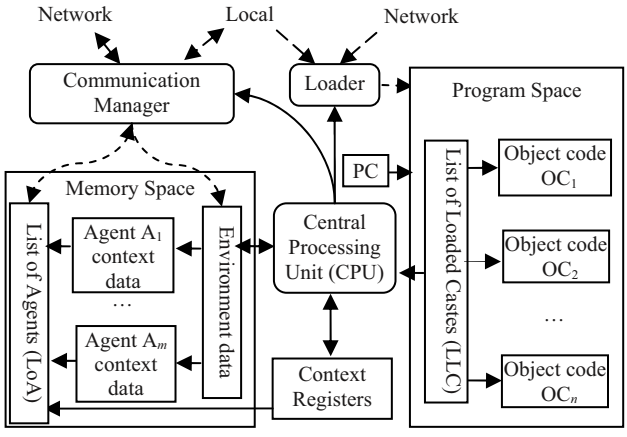


Figure 6. Structure of Local Execution Engine (LEE)

Registers, which is a set of registers that store the context information of the current agent. It includes two parts: the offset of the agent's local variables in memory and the pointer to the agent's operand stack. Within any one particular agent's data in MS, it has its own local variables and operand stack, which are two major runtime data structures used by instructions to store/access runtime intermediate states.

3.3. Communication Engines

The main functionalities of communication engines include deployment management, membership management and communication support of its resident castes. As shown in Figure 7, a communication engine consists of the following components. Receiver and Dispatcher are communication units for receiving/dispatching messages from/to LEEs. Communication Manager (CM) manages the state and action lists of the active agents of the resident castes, according to the environment descriptions, which serves as the definition of the subscriptions, by means of Observe messages sent by LEEs, to the events of observable actions and state changes. Once an observable action or state change is received, the Dispatcher sends the agent's state/action change to the LEE on which those agents who are observing it are running. Membership Manager (MM) manages the activeness status of all the agents of the resident castes, which can be distributed over the network. A data structure, Membership List, is used to trace the activeness. Deployment Manager (DM) manages the deployed castes' object code and Publication Space (PubS) is the memory space that stores the states and actions published by its active agents.

3.4. Interactions between LEEs and CEs

One of the key features of the CAVM is its support to the network transparent communications between the agents. This is achieved by separating LEE and CE. The interactions between LEEs and CEs are realized through the communication messages between LEEs and CEs, which can be classified into the following categories.

Register/Deregister. When one agent is created as an instance of a caste or joins a caste resident on a CE, it registers to the caste through a Register message sent to the host CE. Receiving such a message, the CE's membership and communication managers updates the caste's information and start to provide services to the agent. Similarly, when an agent of a caste is destroyed or quits from the caste, a deregister message is sent to the host CE. Consequently, the CE updates its information and stops the services.

State/Action observe/update. At a high level caste-centric agent-oriented programming language, agents communicate with each other through taking observable actions and updating observable state variables and perceiving other agents' actions and state variables. An agent *A*'s observable actions and variable updating are compiled into instructions that instruct the LEE to send Action or State Update messages to the caste's host CE, which are forwarded to the LEE on which agents that observes agent *A* execute according to their environment descriptions. The environment descriptions are also compiled into instructions that instruct the LEE to send Action or State Observe messages to the corresponding host CE. This is similar to the subscription/publishing mechanism in many middleware, but represented at a higher level of abstraction and implemented with more flexibility.

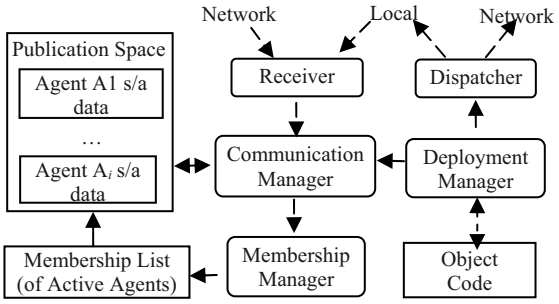


Figure 7. Structure of Communication Engine (CE).

Membership book keeping. The host CE of a caste keeps the track of the aliveness state of its agents, which can also be queried by agents, for example, to obtain a list of live agents of a caste. This is also realized through instructions that results in a message sent to the caste's host CE.

The messages are encoded in XML format. For example, when an agent of caste `GreetingAgent` performs an action `HelloWorld`, an update message is sent to the CE, which in turn automatically propagates the change to the Monitor agent that observes it. When the update message is received by an LEE where a `Monitor` agent runs, the Communication Manager will update the corresponding environment data in its Memory Space.

3.5. Instruction Set

There are three types of CAVM instructions. The *computation* instructions perform computation and local control functions. It includes arithmetic and logic operations as well as control and stack operations. The *interaction* instructions deal with the interactions between agents and castes. It include caste loading, agent's joining and quitting a caste, agent creation and deletion, state update, action event publishing, message sending and receiving, and event publishing and subscription. The *external invocation* instructions are those operations facilitating CAVM's interaction with native environment, such as invocation of third-party runtime libraries (e.g. DLL library) on the host machine, and those for debugging purpose. Details of the instruction set are omitted for the sake of space.

4. Implementation and Evaluation

A prototype system of CAVM is implemented with C/C++ using Visual Studio .NET. LEE and CE are realized as two separate Common Language Runtime console servers. All the functions described in section 3 have been implemented. To facilitate experiments with and evaluation of the design and implementation of the VM, a GUI interface is also developed for the deployment and execution of object code, the measurement of system performance and the management of the VMs running over a network.

Figure 8 is a screen snapshot of the interface. It shows the object code of a caste on the left part of the window. On the right hand side are the IP addresses of the CEs on which object codes are (to be) deployed. The performances of CEs and LEEs are monitored and information is displayed on the tab CE monitor and LEE monitor, respectively. It also provides remote control over agents distributed over a network, such as remote agent launching.

To test the system and evaluate its performance, several preliminary experiments have been conducted. The experi-

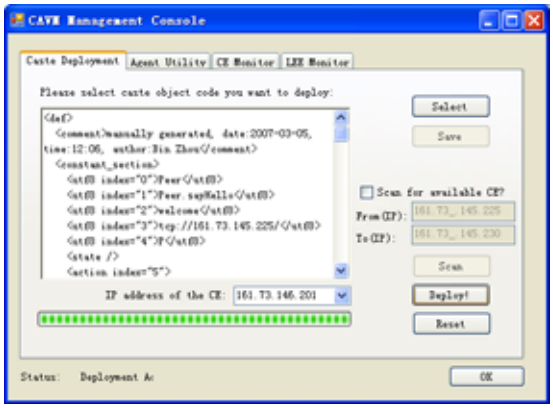


Figure 8. Screen snapshot of the management tool.

ments were performed in a network environment consists of several computers (depending on different experiments). All the computers have Intel Pentium 1.70GHz CPU, 512MB RAM, interconnected by 100M Ethernet. The computation performance is measured by the number of instructions per milli-second (IPmS) and communication performance is measured by the number of messages per milli-second (MPmS).

Figure 9 shows the result of experiments in which the performance of a system that consists of a number of agents communicating to another agent running on the same computer (Exp1) is compared with the performance when the other agent is running on a different computer (Exp2). The performance difference of Exp1 and Exp2 is largely because in the former the communication is via shared memory while the latter is through TCP.

Figure 10 shows the results of experiments in which agents are distributed to a number of computers and each computer hosts 100 agents. The performance of the system only declines slightly due to communication overhead when the number of computers increases.

From the results of the experiments, we can conclude that the design and implementation of the prototype CAVM is efficient in performance and scalable for running a large number of agents over a network.

There are more experiments with the system. Due to the limit on space, the data will be reported in another paper.

5. Related Works

There are two classes of related works. One is the implementation of agent- oriented programming languages and the other is virtual machines.

A few programming languages have been proposed and implemented to directly implement agent-oriented concepts in the literature. In [11], Rafael H. Bordini et al. classified these agent- oriented languages into three categories: purely declarative (e.g. CLAIM [12]), purely imperative (e.g. JACK [13]), and hybrid languages that combines declarative and imperative features (e.g. 3APL [14], Jason [15], and IMPACT [16]). They also surveyed those platforms which realized the semantic of those languages. The CAOPLE is an imperative programming language with language facilities of high level abstractions that directly support the caste-centric approach.

Such implementations have been focusing on the supports to programming in mentalist models of agents such as belief, desire, intension, reasoning, and planning through extending existing programming languages and concepts based on logic or object-oriented languages. However, their programming platforms are either centralized or using a simple distributed computing models (e.g. RMI) to support agent communication in decentralized environment.

The JACK Agent Language (JAL) [13] is probably the most similar to CAOPLE language. It is also an imperative programming language, which extended Java by adding a number of declaration types used to declare agents, beliefsets, views, events, plans and capabilities, and statements to manipulate events in an imperative manner. In addition to the development environment and debugger, JACK's platform contains a light weight communication mechanism to support the sending and receiving messages between agents. The address of the agent on the computer in the form of portal is required when an agent send a message to another, while our VM supports network transparency at high level programming language so that agents can communicate with each other without explicitly specify network address of the agents as shown in our examples. In comparison with JACK, CAVM provides a much higher level communication facility and autonomic mechanism.

In our previous work of experiments with design and implementations of agent oriented programming languages, the SLABSp language [17] is also based on the caste centric model. It extends Java with caste and scenario facilities. The implementation of SLABSp uses components and middleware in a similar way as JACK's platform. It is observed that the VM approach reported in this paper is more flexible and more efficient.

VMs have long been used in hardware virtualization, representing intermediate structures, and bytecode interpretation [18]. They have drawn renewed attention in recent years for their advantages in resource virtualization in the network environments [6]. As a virtual machine of TinyOS, Mate [19] can reprogram the sensor network by sending and receiving messages that enable the deployment of ad-hoc routing and data aggregation algorithm. This feature is similar to CAVM's dynamic loading of object code to LEEs.

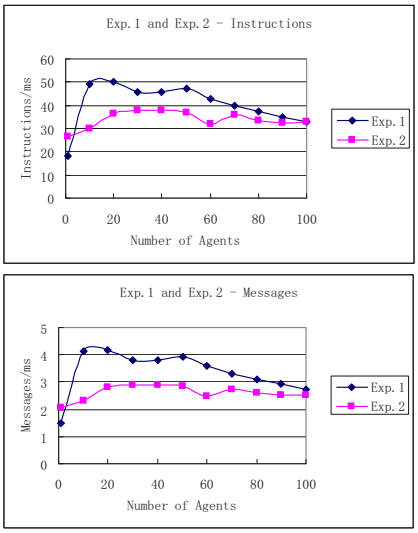


Figure 9. One computer vs two computers

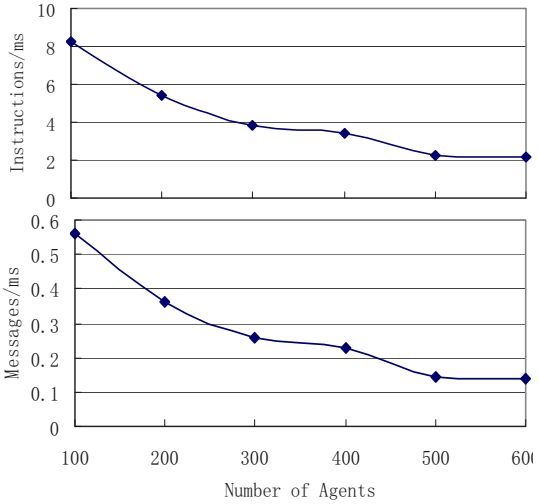


Figure 10. Performance as the numbers of computers and agents increase

Comparing to Mate, CAVM is more powerful, flexible and at a higher level of abstraction. Moreover, by decoupling computation from communication, CAVM enables LEE to be run on less powerful devices while leave communication tasks to be handled by CE running on computers of high network bandwidth and processing power. Of course, the main difference between CAVM and Mate is that CAVM is a high level language VM while Mate is a system VM according to Smith and Nair's classification. A well-known high language level VM is the Java Virtual Machine JVM [20], which provides platform independence to the object oriented programming language Java. Similar to JVM and all other high level language VMs, CAVM provides an abstract layer of indirection for efficient implementation of a high level programming language that is not directly supported by the hardware architecture and instructions. However, unlike JVM, CAVM supports distributed computing from the instruction level rather than using an add-on distributed object model (e.g. RMI). Thus, distributed programs can be written at a higher level of abstraction without being forced to comply with a distributed computation model.

Finally, the publish/subscribe paradigm has the feature of decoupling the communicating parties in time, space and flow, and facilitating concurrent asynchronous computations, which is essential for Internet-based computation. The mechanism has been implemented in various middleware, but few in VMs [21]. The communications between LEEs and CEs in CAVM can be viewed as a publish/subscribe model, but it is in the agent-oriented metaphor described at a high level of abstraction in the form of environment description. In particular, it is unnecessary for an agent to hold each other's references to actively participate in interaction. The built-in communication management mechanism in CAVM enables an asynchronous updates of agents' state/action changes.

6. Conclusion and Future Work

The main contribution of the paper is a virtual machine CAVM designed for the implementation of caste-centric agent-oriented programming language CAOPLE. Its architecture consists of local execution engines (LEEs) and communications engines (CEs) distributed over a TCP/IP network such as the Internet. It provides the mechanisms and facilities to support inter-agent communications with location transparency, dynamic code distribution for agents' dynamic joining to and quitting from castes and creating agents as instances of castes whose object codes are deployed to computers on the network. Experiments with the performance of the VM were reported, which demonstrated the efficiency and scalability of the system.

Currently, we are completing a compiler that translates CAOPLE source code to the CAVM object code. We are also developing a library of graphic user interface agents for dynamic construction and adaptation of graphic user interfaces. Finally, case studies with real applications are also on our agenda.

Acknowledgments

The authors would like to thank the Applied Formal Method research group at Oxford Brookes University for discussions on related topics and comments on the earlier draft of the paper. The work is partially supported by China Ministry of Science and Technology in the National Basic Research Program (973 program) under the grant 2005CB321800. The work is carried out while the first author, Bin Zhou, was visiting Oxford Brookes University.

References

[1] Jennings, N. R.: On agent-based software engineering. *Artificial Intelligence* 117, 277–296, 2000.

[2] Jennings, N.R., Wooldridge, M.J. (eds.): *Agent Technology: Foundations, Applications, and Markets*. Springer, 1998.

[3] Zambonelli, F. and Omicini, A.: Challenges and Research Directions in Agent-Oriented Software Engineering, *AAMAS* 9, 253-283, 2004.

[4] Henderson-Sellers, B. and Giorgini, P. (Eds.): *Agent-oriented Methodologies*, Idea Group Publishing, June 28, 2005.

[5] Padgham, L., and Zambonelli, F.: *Agent-Oriented Software Engineering VII,* LNCS 4405. Springer, 2007.

[6] Figueiredo, R., Dinda, P. A., Fortes, J.: Resource Virtualization Renaissance, *Computer* 38(5), 28-31, 2005.

[7] Zhu, H.: SLABS: A formal specification language for agent based systems. *SEKE* 11(5), 529–558, 2001.

[8] Zhu, H. and Shan, L., Caste-Centric Modelling of Multi-Agent Systems: The CAMLE Modelling Language and Automated Tools, in *Model-driven Software Development*, Beydeda, S. and Gruhn, V. (eds), 57-89. Springer, 2005.

[9] Zhu, H.: Towards An Agent-Oriented Paradigm of Information Systems. *Handbook of Research on Nature Inspired Computing for Economy and Management*, Jean-Philippe Rennard (Ed), Idea Group Inc. Chapter XLIV, 679–691, 2006.

[10] Mao, X., Shan, L., Zhu, H and Wang, J.: An Adaptive Casteship Mechanism for Developing Multi-Agent Systems, *Intl. J. of Computer Application in Technology*. (In press)

[11] Bordini, R. H., et al.: A survey of programming languages and platforms for multi-agent systems. *Informatica* 30(1), 33-44, 2001.

[12] Seghrouchni A., Suna. A.: CLAIM: A computational language for autonomous, intelligent and mobile agents. *Programming Multiagent Systems*, LNCS 3067, Springer Verlag, 2004.

[13] Winikoff. M.: JACKTM intelligent agents: An industrial strength platform. In *Multi-Agent Programming: Languages, Platforms, and Applications*, Bordini, R. H., Dastani, M., Dix, J., Seghrouchni, A. (eds.), Springer, Chapter 7, (2005)

[14] Hindriks, K., de Boer, F., van der Hoek, W., Meyer. J.: Agent programming in 3APL. *AAMAS* 2(4), 357–401, 1999.

[15] Rao. A. S.: AgentSpeak(L): BDI agents speak out in a logical computable language. *Proc. of Modelling Autonomous Agents in a Multi-Agent World*, LNAI 1038, 42–55. Springer, 1996.

[16] Subrahmanian, V., et al.: *Heterogenous Active Agents*. MIT-Press 2000.

[17] Wang, J., Shen, R., Zhu, H.: Agent oriented programming based on SLABS. *Proc. of COMPSAC'05*, 127–132, 2005.

[18] Smith, J. E. and Nair, R.: The Architecture of Virtual Machines, *Computer* 38(5), 32-38, 2005.

[19] Levis P., Culler. D.: Mate: A tiny virtual machine for sensor networks. *Proc. of Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2002.

[20] Lindholm, T., Yellin, F.: *The Java Virtual Machine Specification*. Second Edition, Addison-Wesley, 1999.

[21] Deng, Y., Sadjadi, S. M., Clarke, P. J. Zhang, C., Hristidis, V., Rangaswami, R., and Prabakar, N.: A Communication Virtual Machine, *Proc. of COMPSAC'06*, 521-531, 2006.

MAAEM: A Multi-agent Application Engineering Methodology

Adriana Leite, Rosario Girardi, and Uiratan Cavalcante
UFMA – Federal University of Maranhão
Av. dos Portugueses, s/n
São Luís, Brazil
{adri07lc@gmail.com, rgirardi@deinf.ufma.br, uiratan@gmail.com}

Abstract

Multi-agent Application Engineering is a process for the construction of applications based on the reuse of agent-oriented software artifacts. This paper introduces MAAEM ("Multi-agent Application Engineering Methodology"), an ontology-driven methodology for multi-agent system development which promotes the reuse of software artifacts previously produced in a Multi-agent Domain Engineering process.

1. Introduction

Domain Engineering, also called development *for* reuse, is a process for the creation of software abstractions reusable on the development of an application family in particular domain problem. Application Engineering or development *with* reuse is a process for the construction of a specific application through the reuse of available abstractions from Domain Engineering.

MAAEM is a methodology for Multi-agent Application Engineering approaching the analysis, design and implementation of multi-agent applications through the reuse of software artifacts previously developed with MADEM [8], a methodology for Multi-agent Domain Engineering. MAAEM approaches the selection, adaptation and composition of these artifacts for the construction of a specific application of the family.

ONTORMAS [10] is an ontology-based tool that conceptualizes the MADEM and MAAEM methodologies and specifies all the guidelines for Multi-agent Domain and Application Engineering.

Infonorma [4] is a multi-agent legal recommender system that recommends legal normative instruments to users according to their particular interests. The system was modeled under the guidelines of MAAEM and this experience has contributed for its evaluation.

This paper introduces the MAAEM methodology, illustrating its application through the development of the Infonorma system.

The paper is organized as follows. Section 2 presents the MAAEM methodology along with some examples of its application on the development of the Infonorma multi-agent system. Section 3 summarizes the ONTORMAS support to the MAAEM methodology. Section 4 presents related work on multi-agent development. Finally, section 5 concludes the paper with some considerations about further work on MAAEM and ONTORMAS.

2. The MAAEM Methodology

For the specification of a problem to be solved, MAAEM focuses on modeling concepts, goals, roles and interactions of entities of an organization.

Entities have knowledge and use it to exhibit autonomous behavior. An organization is composed of entities with general and specific goals that establish what the organization intends to reach. The achievement of specific goals allows reaching the general goal of the organization. For instance, an information system can have the general goal of "satisfying the information needs of an organization" and the specific goals of "satisfying dynamic or long term information needs". Specific goals are reached through the performance of responsibilities in charge of particular roles with a certain degree of autonomy.

Roles have skills on one or a set of techniques that support the execution of responsibilities in an effective way. Pre-conditions and post-conditions may need to be satisfied for/after the execution of a responsibility. Knowledge can be consumed and produced through the execution of a responsibility. For instance, an entity can play the role of "retriever" with the responsibility of executing the responsibilitty of satisfying the dynamic information needs of an organization. Another entity can play the role of "filter", in charge of the responsibility of satisfying the long-term information needs of the organization. Skills can be, for instance, the rules of the organization to access and structure its information sources. Sometimes, entities have to communicate with other internal or external entities to cooperate in the execution of a responsibility. For instance, the entity playing the role of "filter" may need to interact with a user (external entity) to observe his/her behavior in order to infer his/her profile of information interests. For the specification of a design solution, roles are assigned to agents structured and organized into a particular multi-agent architectural solution according to non-functional requirements.

The tasks performed, the resources required and the products obtained at each phase of the MAAEM methodology are described in the following sections an summarized in Table 1.

Table 1 - Phases, Resources, Tasks and Products of the MAAEM methodology

Phases	Resources	Tasks		Products		
Application Requirements Engineering	Concept Model of MADEM	Concept Modeling		Concept Model		Application Specification
	Knowledge of Specialists, Literature in the Domain, Existing applications of Software					
	Goal Model of MADEM	Goal Modeling		Goal Model		
	Application Family Requirements					
	Role Model of MADEM	Role Modeling		Role Model		
	Goal Model					
	Role Interactions Models of MADEM	Role Interactions Modeling		Role Interactions Models		
	Goal Model					
	Role Interactions Models					
	Role Model					
	User Interface Prototype of MADEM	User Interface Prototyping		User Interface Prototype		
	Role Interactions Models					
	Role Model					
Application Design	Multi-agent Society Knowledge Model of MADEM	Architectural Design	Multi-agent Society Knowledge Modeling	Multi-agent Society Knowledge Model	Architectural Model	Application Architecture
	Concept Model					
	Role Interaction Model					
	Multi-agent Society Model of MADEM		Multi-agent Society Modeling	Multi-agent Society Model		
	Role Model					
	Agent Interaction Model of MADEM		Agent Interaction Modeling	Agent Interaction Model		
	Multi-agent Society Model					
	Activity Models of MADEM		Activity Modeling	Activity Models		
	Multi-agent Society Model					
	Agent States Models					
	Coordination and Cooperation Model of MADEM		Coordination and Cooperation Modeling	Coordination and Cooperation Model		
	Goal Model (Non-functionais Requeriments)					
	Multi-agent Society Model					
	Agent Interaction Model					
	Agent Knowledge Models of MADEM	Agent Design	Agent Knowledge Modeling	Agent Knowledge Models	Agent Models	
	Role Interactions Models					
	Multi-agent Society Knowledge Model					
	Agent State Models of MADEM		Agent State Modeling	Agent State Models		
	Multi-agent Society Model					
	Agent Interaction Model					
	Activity Models					
Application Implementation	Model of Behaviors of MADEM	Behaviors Modeling		Behaviors Model		Application Implementation Model
	Activity Models					
	Model of Communication Acts of MADEM	Communication Acts Model		Communication Acts Model		
	Agent Interaction Model					
	Executable software agents of MADEM	Agent Implementation		Executable Software Agents		
	Model of Behaviors					
	Model of Communication Acts					

2.1. The Application Analysis Requirements Phase

The MAAEM requirements analysis phase is performed through the following modeling tasks: concept modeling, goal modeling, role modeling, role interaction modeling and user interface prototyping. This phase intends to identify and specify the requirements of a particular application from domain models already created in the respective phase of the MADEM methodology [8]. Thus, the central task of a developer is to reuse a set of requirements of an application family in a domain through their selection among the common and variable requirements in a domain model.

The modeling of domain application concepts task aims at performing a brainstorming of domain concepts and their relationships, representing them in a *Concept Model*. These concepts are refined in the subsequent modeling tasks.

The purpose of the goal modeling task is to identify the goals of the family of systems, the external entities with which it cooperates and the responsibilities needed to achieve them. Its product is a *Goal Model*, specifying the general and specific goals of the system family along with the external entities and responsibilities. Figure 1 illustrates the Goal Model of Infonorma.

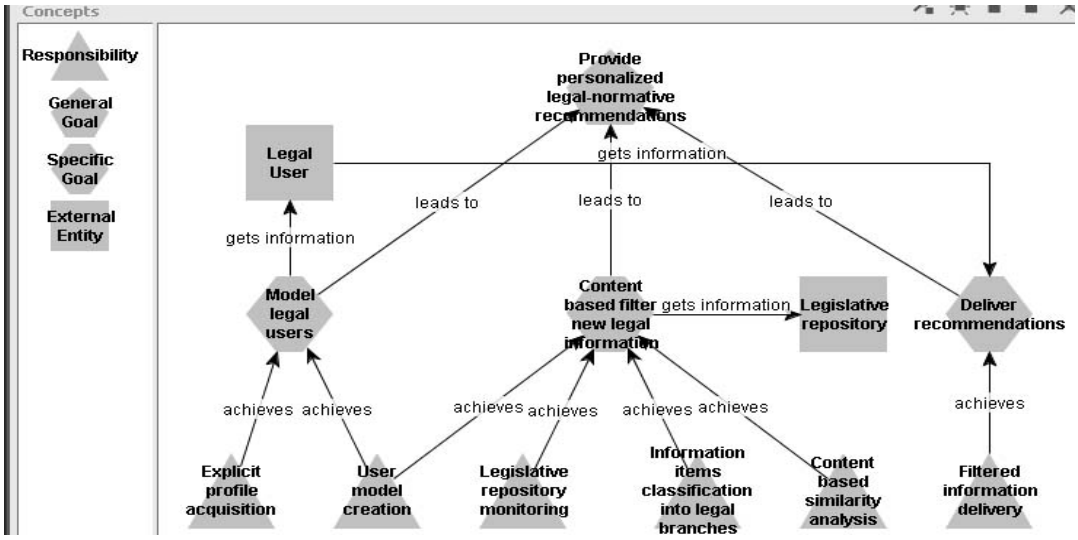


Figure 1 – Goal Model of Infonorma

The role modeling task associates the responsibilities identified in the goal modeling task to the roles that will be in charge of them. The pre and post-conditions that must be satisfied before and after the execution of a responsibility are also identified. Finally, the knowledge required from other entities (roles or external entities) for the execution of responsibility and the knowledge produced from their execution is identified. This task produces a *Role Model*, specifying roles, responsibilities, pre- and post-conditions, knowledge and relationships between these concepts.

The role interaction modeling task aims at identifying how external and internal entities should cooperate to achieve a specific goal. For that, responsibilities of roles are analyzed along with their required and produced knowledge specified in a role model. A set of *Role Interaction Models*, one for each specific goal in the *Goal Model*, specifying the interactions between roles and external entities needed to achieve a specific goal is constructed as a product of this task.

Finally, is constructed a *User Interface Prototype*, identify the interactions of the users with the system and simulate them in a prototype.

2.2. Application Design Phase

In the Application Design Phase developers reuse design solutions of a family of applications and adapt them to the specific requirements of the application under development. Otherwise, they design a multi-agent architecture from scratch, providing a solution to the requirements of the multi-agent application specified in Analysis Requirements Phase.

This phase consists of two tasks: the Architectural Design task aiming at constructing a multi-agent society architectural model and the Agent Design task, which defines the internal structure of each agent in the society.

The Architectural Design task consists of five sub-tasks: Multi-agent Society Knowledge Modeling, Multi-Agent Society Modeling, Agent Interaction Modeling, Activity Modeling, and Coordination and Cooperation modeling. The purpose of the multi-agent society knowledge modeling subtask is to represent the meaning of concepts that agents of the society need to understand in order to communicate with each other. This is done through the construction of a model of the multi-agent society knowledge, represented in a semantic network. In the Multi-Agent Society Modeling sub-task, the roles identified in the application analysis phase are assigned to agents. An agent can play one or more roles according to the affinity between their responsibilities, number of interactions between them or functional cohesion criteria. In the Agent Interaction Modeling sub-task, the interactions between the agents are specified. Messages are specified following the FIPA-ACL guidelines. The Activity Modeling sub-task aims at detailing the activities carried out to achieve the specific goals. Its product is a set of activity models, one for each specific goal of the application. The Coordination and Cooperation modeling sub-task aims at either reusing or creating appropriate mechanisms of coordination and cooperation between agents to produce a coordination and cooperation model satisfying non-functional requirements. Figure 2 illustrates part of the Multi-agent Society Model of Infonorma showing the specification of the Filter agent.

The Detailed Design task is constituted of two sub-tasks: Agent Knowledge Modeling and Agent State Modeling. The Agent Knowledge Modeling sub-task specifies the knowledge of each agent. It is built based on the agent knowledge previously specified on the agent activity models and on the multi-agent society knowledge model. The Agent State Modeling sub-task aims at representing the states that the agent assumes during its lifecycle, as well as the transitions that lead from one state to another. The activities identified in the activity model are associated with the states identified here. For each state it is specified the activity that the agent performs when it assumes the state, namely entry activity, the actions, which is a set of activities that can be performed in the current state and the exit activity, which is the one performed right before the agent leave its current state.

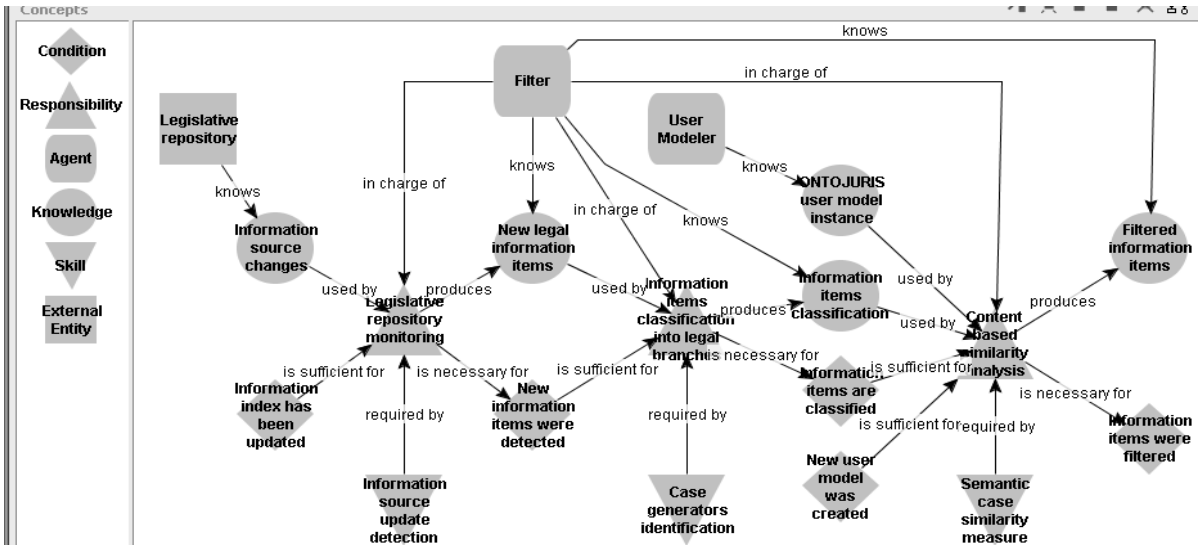


Figure 2 - Multi-agent Society Model of Infonorma showing the Filter agent

2.3. Application Implementation Phase

In the Application Implementation Phase, agents are identified from the Activity Models of the previous phase. A behavior is associated to each responsibility in a particular language/platform of agent's development, such as JADE [1], producing a Behaviors Model. Agent's interactions are identified in the Agent's Interactions Model, which are mapped to a specific language for communication between agents, as FIPA_ACL [4], resulting in a Communication Acts Model. In this phase, the selection of agents for reuse occurs based on the similarity of behaviors and agent communications. Figure 3 illustrates the Communication Acts Model of Infonorma, showing the communication between the Filter and User Modeler agents.

3. Multi-agent Application Engineering Support

ONTORMAS [10] is an ontology-based tool developed in Protégé environment [6] that provides support to the modeling tasks of both MADEM and MAAEM methodologies. Modeling products are generated through the instantiation of the corresponding subclasses of the *Modeling Tasks*, *Modeling Products*, and *Modeling Concepts* classes of the ONTORMAS ontology. Once instantiated, ONTORMAS becomes a knowledge base where concepts in modeling products are semantically related, and, therefore, queries and inferences through logical reasoning can be done to facilitate the understanding and reuse of modeling products.

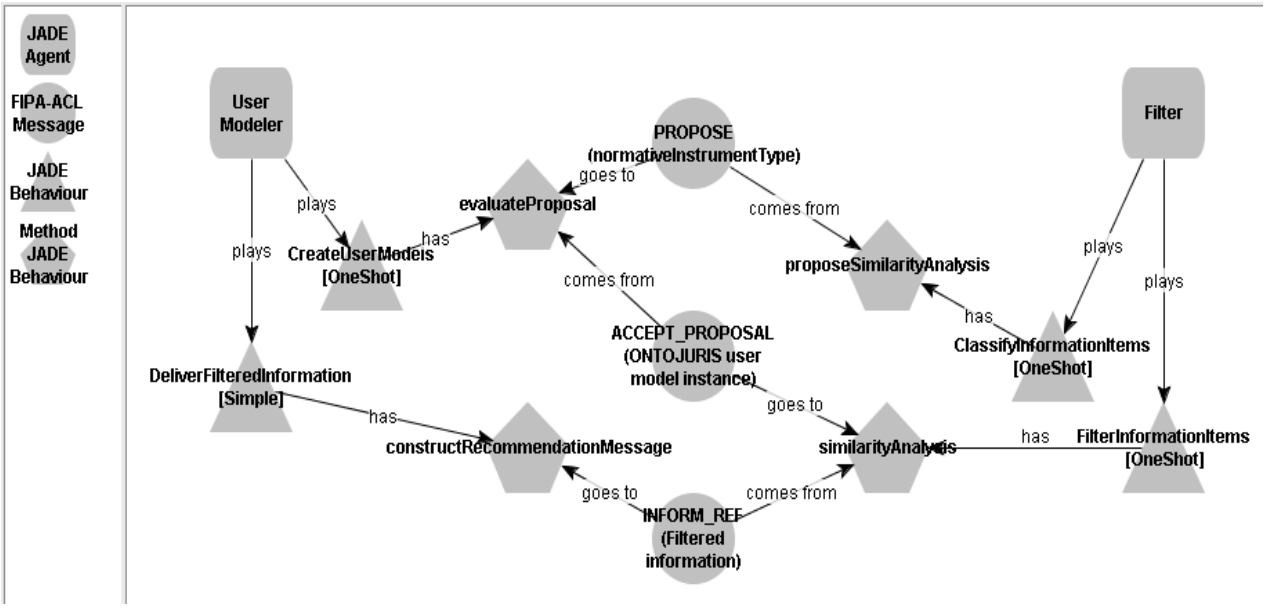


Figure 3 - Communication Acts Model of Infonorma

Figure 4 illustrates some semantic relationships between classes and instances in the ONTORMAS knowledge base.

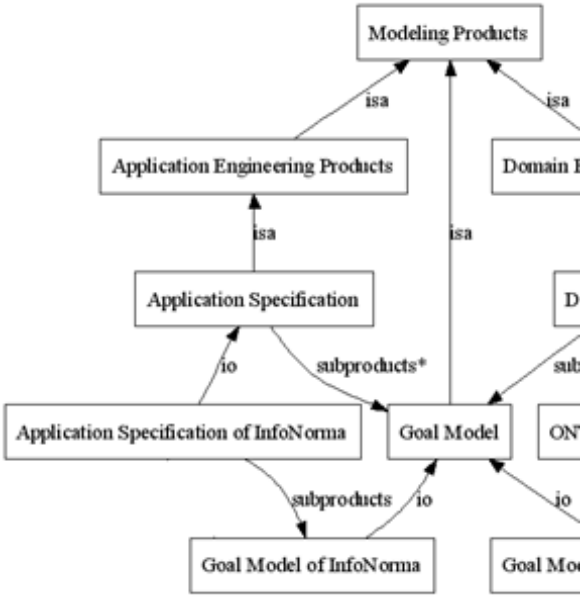


Figure 4 – Some semantic relationships between classes and instances int the ONTORMAS knowledge base.

Since retrieval is based on semantics, results from searching on modeling products and concepts in the ONTOMADEM knowledge base are more effective than the ones that could be obtained through simple keyword retrieval on instance texts. After the selection of the artifact that most closely matches their needs, a user should check if the artifact can be integrally reused or if it needs adaptation or integration with other artifacts.

Also, a graphical notation has been defined in ONTORMAS for the representation of each modeling product. This facilitates not only the instantiation process but also contributes for reducing the complexity of the modeling tasks, allowing the visualization, decomposition and refinement of the modeling products.

4. Related Work

Some methodologies, like GAIA [11], PASSI [2] and TROPOS [7], have been already developed to increase the productivity of the software development process, the reusability of generated products, and the effectiveness of project management.

GAIA is a methodology based in human organization concepts. It supports the analysis and design phases for multiagent system development. Tropos is an agent-oriented software development methodology supporting the complete multi-agent development process. It is based on the i* organizational modeling framework. PASSI is a methodology for multi-agent development integrating concepts from object-oriented software engineering and artificial intelligence approaches. It allows the development of multi-agents systems for special purposes as mobiles and robotics agents and uses an UML-based notation. There are available tools for PASSI and TROPOS modeling and code generation. However, only PASSI allows code reuse.

Two main features distinguish MAAEM from other existing approaches. First, it provides support for reuse on multi-agent software development, through the integration of the concepts of Domain Engineering and Application Engineerings. Second, it is a knowledge-based technique where models of agents and frameworks are represented as instances of the ONTORMAS ontology. Thus, concepts are semantically related allowing effective searches and inferences thus facilitating the understanding and reuse of the models during the development of specific applications in a domain.

5. Conclusions and further work

This paper described MAAEM, an ontology-based methodology for analysis, design and implementation of multi-agent systems. The software artifacts produced by MAAEM are represented as instances of the ONTORMAS ontology, which serves as a repository for reusable software artifacts and also as a tool supporting application development.

MAAEM is a part of a project for the improvement of multi-agent system development techniques, methodologies and tools. With the knowledge base provided by ONTORMAS, an expert system is being developed, aiming at automating various tasks of both MADEM and MAAEM, thus allowing fast application development and partial code generation.

MAAEM currently supports compositional reuse, based on the selection, adaptation and composition of artifacts. A generative approach [3] for reuse has been explored with the specification of the GENMADEM methodology and the ONTOGENMADEM tool [9]. ONTOGENMADEM provides support for the creation of Domain Specific Languages to be used on the generation of a family of applications in a domain. Further work will extend ONTORMAS for supporting ONTOGENMADEM allowing generative reuse in Multi-agent Applications Engineering.

References

[1]BELLIFEMINE, F., CAIRE, G., POGGI, A., RIMASSA, G.. (2003). JADE A White Paper. Exp v. 3 n. 3, Sept 2003. http://jade.tilab.com/

[2]COSSENTINO, M., SABATUCCI, L., SORACE, S. and CHELLA, A. (2003). Patterns reuse in the PASSI methodology. *In: Proceedings of the Fourth International Workshop Engineering Societies in the Agents World (ESAW'03)*, pp. 29-31. Imperial College London, UK.

[3]CZARNECKI, K., EISENECKER, U. W. (2000). Generative Programming: Methods, Tools, and Applications. ACM Press/Addison-Wesley Publishing Co., New York, NY.

[4]DRUMOND, L. and GIRARDI, R. (2008). A Multi-agent Legal Recommender System. *Journal of Artificial Intelligence and Law*, (to appear).

[5]FIPA. ACL Message Structure Specification. Available at:

<http://www.fipa.org/specs/fipa00061/SC00061G.html>. Accessed in: March 02 2008.

[6]GENNARI, J., MUSEN, M. A., FERGERSON, R. W. et al. (2002). The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. Technical Report SMI-2002-0943.

[7]GIORGINI, P., KOLP, M., MYLOPOULOS, J., & Pistore, M. (2004). The Tropos methodology: An overview. *In Methodologies and software engineering for agent systems*. Boston: Kluwer Academic Publishing, pp. 89-106.

[8]GIRARDI, R. and BALBY, L. (2007). A Domain Model of Web Recommender Systems based on Usage Mining and Collaborative Filtering. *Requirements Engineering* , v. 12, n. 8, pp. 23-40.

[9]JANSEN, M., GIRARDI, R. (2006). GENMADEM: A Methodology for Generative Multi-agent Domain Engineering. In: The 9th International Conference on Software Reuse, 2006, Torino. *Proceedings of the 9th International Conference on Software Reuse*, Lecture Notes in Computer Science (LNCS), v. 4039, p. 399-402. Berlin: Springer-Verlag.

[10]LEITE, A. GIRARDI, R. (2008). ONTORMAS: An Ontology-driven tool for Domain and Application Engineering, *The XI Iberoamerican Workshop on Requirements Engineering and Software Environment*. (In Portuguese, to appear).

[11]ZAMBONELLI, F., JENNINGS, N., and WOOLDRIDGE, M. (2003). Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, pp. 417-470.

A Semantic Based Certification and Access Control Approach Using Security Patterns on SEAGENT

Fatih Tekbacak
Department of Computer Engineering
Izmir Institute of Technology
Urla, Izmir, Turkey 35430
Email: fatihtekbacak@iyte.edu.tr

Tugkan Tuglular
Department of Computer Engineering
Izmir Institute of Technology
Urla, Izmir, Turkey 35430
Email: tugkantuglular@iyte.edu.tr

Oguz Dikenelli
Department of Computer Engineering
Ege University
Bornova, Izmir, Turkey 35100
Email: oguz.dikenelli@ege.edu.tr

***Abstract*—In this paper, we propose a security infrastructure for communication between agents adaptable to FIPA security specifications by employing security patterns and semantic based policy descriptions. Security patterns are used as a generalized approach for generating security based services. This paper analyzes the authentication and semantic based access control among agents by using the security patterns.**

I. Introduction

Multi-agent systems(MAS), which communicate in an open environment like Internet, face safety and security problems. Therefore, MAS should have some strategies, policies and mechanisms for confidentiality, integrity, authentication, non-repudiation [1], [2], [3]. This paper proposes a security infrastructure intended for a FIPA compliant multi-agent system namely SEAGENT [4], and provides a solution by using security patterns with semantic web for policy based approaches. SEAGENT agents demanding for secure access will be utilizing *Agent Authenticator*, *Agent Certification Authority(ACA)* and *Access Controller* patterns that have been explained in Tropos methodology [5].

II. Background

SEAGENT is a P2P Java framework for writing Semantic Web enabled Multi-Agent applications [4]. The main objective of SEAGENT project is to develop an agent framework for constructing FIPA-compliant multi-agent platforms that work on semantic web environment. It's communication module supports current web based communication protocols both for intra-platform and inter-platform communication.

Agent oriented software engineering is one of the most natural ways of characterizing security issues in information systems. This approach allows developers first to model the security requirements in high-level and then incrementally apply these requirements to security mechanisms as services(or agents) in the multi-agent systems [6], [7].

This paper uses the approach that has been detailed in [7]. The authors of Tropos merged the advantages of the agent oriented programming and security patterns paradigms by applying both of them in the Tropos methodology. Secure Tropos extends the agent oriented software engineering methodology by providing a set of security-related concepts and processes to allow developers to consider security issues throughout the development stages. By using this methodology, agent oriented concepts could identify a set of security requirements needed by the system and these requirements can be transformed to a design with the use of security patterns.

Since SEAGENT is a FIPA-based multi-agent system, FIPA specifications have been followed throughout this work. First, all agents must register to Agent Management System(AMS). AMS has the responsibility to monitor the lifecycle of agents and agents must inform AMS about their platform related actions(register, deregister and so on). Second, software agents must also register their service descriptions to Directory Facilitator(DF). During this process, a masquerading agent should be prevented from registering its services or service descriptions and at the end damaging the platform. Third, there is also a communication layer called Agent Communication Channel(ACC) which transmits agent communication messages. Those messages should have confidentiality, integrity, authentication and non-repudiation properties according to FIPA security specification. This specification introduces Agent Platform Security Manager(APSM) which defines security issues of AMS and requires a PKI for registering agents. The specification of FIPA for message-based communication security uses Agent Communication Language(ACL) envelope added properties.

III. Security Patterns

Security is often an afterthought functionality in system design and implementation. The enterprise context and requirements that drive system security are often not addressed explicitly and are not incorporated into system architectures. The desired approach is to begin to address security together with the system design rather than the repair-service approach [3].

The basic idea behind patterns is to capture expert knowledge in the form of documentation with a specific structure containing proven solutions for recurring problems in a given domain. In particular, security patterns can be more useful when people responsible for systems have little or no security expertise.

In this paper, *Agent Authenticator*, *Agent Certification Authority* and *Access Controller* patterns have been examined in detail. The remaining patterns defined in [7], namely *Agency Guard* and *Sandbox* patterns are out of scope of this paper.

A. Agent Authenticator

Agents must be authenticated in the platform before they are allowed for intra- and inter-platform communication. The *Agent Authenticator* pattern determines the authentication process of agents in an agency. The authentication process is implemented by using digital signatures generated with agent's or agency's secret key.

The advantage of *Agent Authenticator* is to check the agent's identity before it involves in any communication within the agency. Authentication of the requesting agent could be verified by *Agent Authenticator*. This pattern also prevents implementation of different authentication mechanisms for different agents.

The disadvantage of this approach is that *Agent Authenticator* becomes a central point. So that when the *Agent Authenticator* crashes, the whole system would be under risk.

The design of the agent authentication model in SEAGENT by using SEAGENT Plan Editor is shown in Figure 1. *SupplyPrivateInformation* behaviour takes the owner policy and key pair of the agent. Outcome of this behaviour is passed to *SupplyPrivateKey* action. Private key of the agent is used for creating its digital signature for authentication issues in *SupplyDigitalSignature* action. *AuthenticationManager* action is the connection point of *AgentAuthenticator* with *AgentCertificationAuthority* to obtain the related request parameters by the issued certificate of the agent. These parameters could be subject, issuer of the certificate, validity time of the certificate to validate the digital signature and apply the authentication rules to decide agent's authentication for the communication. *AgentAuthenticator* lastly takes its external provision as RequestParameters from *AgentCertificationAuthority* and passes these parameters to *AuthenticationManager* by provision inheritance. So the Agent Authentication mechanism halts by the decision of this behaviour's planning activities and the outcome of *AuthenticationManager* causes the authentication decision for the requestor agent.

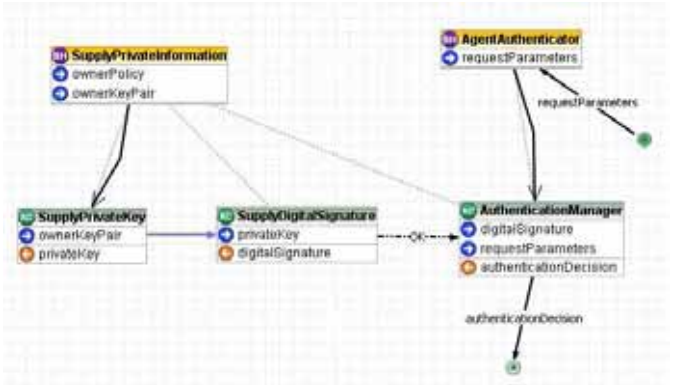


Fig. 1. Agent Authenticator Plan in SEAGENT (General View)

B. Agent Certification Authority

In a trusted environment, every agent is required to possess a certificate which includes a public key. The corresponding private key is stored by the agent in a secure manner. These agents verify their identities by generating digital signatures using their own private keys.

The advantage of *Agent Certification Authority* is the ability to verify a requestor agent. So that the indicated public key is proven to be really used for the communication. This pattern helps to design an appropriate signature verification mechanism to satisfy identity and authentication requirements for a specific domain or situation [3]. The disadvantage of this pattern could be scalability problem when a lot of agents want to request for certification.

C. Access Controller

This is a pattern that restricts agent access to resources. Agents with various privileges can exist in an agency. Agents requesting for a resource could be denied or accepted according to the requested action. Agents in the agency could access the resources(or other agents) according to the response of *Access Controller*. These responses have been sent to the agents with the indicated privileges. If there is an agent's resource requirement instead of access to an agent, *Resource Manager* behaves as a helper service to the *Access Controller* and accesses the related agents' resources.

The advantage of *Access Controller* is the usage of different policies for different actions. In Figure 2, the ACA gives *Agent2* to *send-to* privilege for communicating with Agent1. These privilege types could be obtained from FIPA-ACL based message envelope by different SecurityObjects. If the *Agent2* tries to take *send-to* privilege, the SecurityObjects for different agents could determine the acceptance or denial of the message with inform or refuse communicative acts in FIPA. There is a disadvantage of this pattern that the crash of the *Access Controller* makes the access protocol unusable in agency.

```
:envelope (
  :destination(...)
  :return-address (...)
  :Security(
     : type authorize
     : permission message
     : policy send-to
     : user Agent1)
  :confidentiality high
  :integrity high)
:message
(inform
  :sender    SCA @seagent
  :receiver Agent2@seagent
  :ontology ....
  :content .....)
```

Fig. 2. FIPA ACL Message Example with Security Access Information

IV. Software Agent Certification

The proposed approach attempts to employ security patterns for model driven design with reusable code and suggests utilization of semantic data on certification and policy based agent access models. It also defines a message extension for a new element that describes a form of the message security.

Essential security services explained in [8] are presented in layers.

ACA is a security wrapper for the system that dominates the protocol steps and supplies them to the agents. During Agent Certification Authority [5] process time, ACA enables both sender and receiver agents to negotiate security parameters and then on agents will communicate using the negotiated values. This negotiation also helps to decouple the multi agent system from selected security approach.

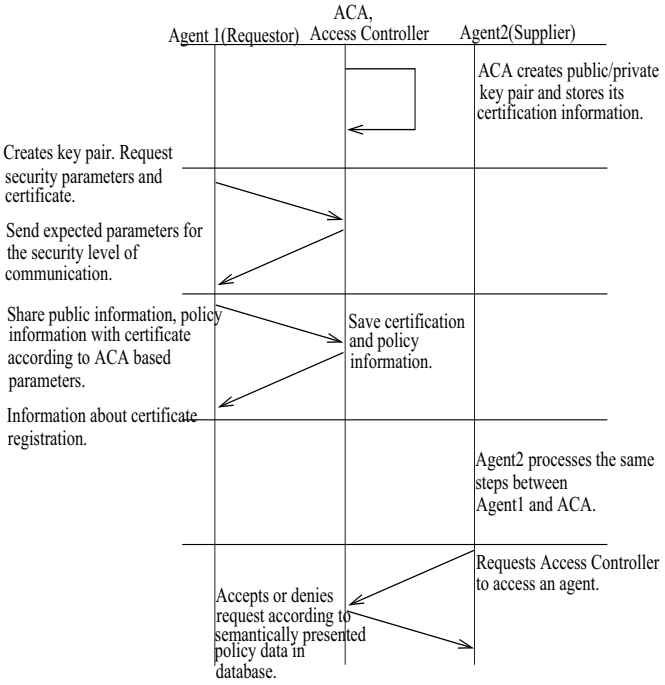


Fig. 3. Steps of Agent Certification Mechanism

The software agent certification steps have been illustrated in Figure 3. First, ACA creates public/private key pair and stores its certification information(issuer, subject, owner, validity, public key with key algorithm information, signature information, default certificate policies and policy mappings). The storage process is based on XML data for syntactic data compatibility and used by all agents. By default, authentication across agents will be accomplished by the same algorithm as the intra-platform communication. This authentication mechanism will be based on ACA as mandatory authentication and access control approach between agents. Because the access information and related certification information for supplier agents have to be supplied to the requestor agents by ACA.

If ACA accepts the requests of the *Agent1*, it sends expected parameters for the security level of communication. ACA accepts the certificate and registration information with policy data from *Agent1* and ACA saves certification information in XML and policy information in OWL(Web Ontology Language) format to its database. Then it informs the requestor agent. The semantic data for policy information aims to collect the policies in a tractable way by the Access Controller. ACA shall contractually require that the subscriber indicates acceptance or rejection of the certificate following its issuance, in accordance with procedures established by the ACA.

Agent1 creates its key pair and stores them as explained in the first step of ACA. Then this agent requests security parameters and certificate from ACA. All private keys and other security related data have to be available to their owner only. Data may not be accessible to other agents (even the agent platform). Every agent has to keep its private data secured but the platform based public information with certificate could be shared between agents according to ACA based parameters. So ACA will send certification information to the agents if the requesting side has the right to communicate in the platform. The certificate policies for agents are initialized in ACA by the security engineer of the system.

Agent1 prepares certificate and requests to register it with semantic policy information. Then the communication of *Agent1* with ACA ends for the certification and semantic data exchange.

The access information have been stored as semantic information shown in Section V. This access information is sent and received with a SecurityObject. The SecurityObject can be included as user-defined slot into the envelope (e.g. X-Security) as used in [8], or, if standardized by FIPA, as an optional slot (e.g. Security). Furthermore, the slot containing the SecurityObject can contain a set of SecurityObjects, for different security attributes applied to a message. The approach told in this paper as adding Security slot to the message Envelope.

ACA accepts or denies this request according to semantically presented data in its database. Information message from ACA to *Agent2* with security slot for access information in a FIPA-ACL message example similar to [9] is shown in Figure 2.

V. SEMANTIC BASED AGENT ACCESS CONTROL MECHANISM

After defining agent certification process details of *Access Controller* mechanism have been explained in this section. First, access control policy includes a set of rules that associate some credentials to use capability of a right. So that the issues with the specified credentials could supply the capability. Credential is any property associated with an entity. When the entity is suitable to the policy rules in the system, the required action for this entity could be populated [10].

All agents have to digitally sign all service requests for AMS, DF and other agents. As there is a public/private key pair for each of the agent, the agent can be thought as accountable by the platform. So that when an agent wants to register to the platform, its credentials have to be checked by *Access Controller* of which decision is based on security policies that have been defined in the platform. These policies could be defined in two levels: platform level and agent level [11]. Platform level policies control the requests for AMS and DF of the platform. Agent level access policies specify who can access the services of the specified agent. A simple message that assigns a right to an agent is shown in Figure 2.

When the certification steps as explained in Section IV have been constructed, AMS controls the validity of the certificate by the default certificate authority in the platform. For organization wide certification, certification path could be chained and the verification step could be processed by the help of *Agent Certification Authority*. If the certificate is valid, AMS restores agent's policies that have to be checked during the communication with AMS. All of these policies are in *Access Controller*'s database by default. AMS and *Access Controller* always communicates with each other to inform the changing policies in the autonomous environment. So when *Access Controller* has been crashed for a short time, AMS based policy rules could still be applied by AMS with its internal policy engine. AMS and DF have a list of conditions that an agent must satisfy in order to contact a particular agent or use a particular service. While AMS and DF have to know the access privileges of agents in [11], *Access Controller* mechanism have to access and know their rights. So that the protection from the threat could be applied in a central place.

Agents could register their services in open or private ways. In open way, the only DF based policies have to be applied for the agents' access to the service. In private way, *Access Controller* communicates with DF and access control mechanism have been processed for the owner of the service. For the verification of rights, a service agent expects all the credentials from the requestor agent at the time of the request in order to use its services. The service agent will check its internal knowledge base and ask for *Access Controller* to give permission to the requestor agent. According to the allowance of the semantic policy information, request could be answered in a positive way. Otherwise, request would be denied until the requestor agent has the suitable credentials to call this service.

VI. Conclusion

In this paper, security patterns that have been used in multi-agent systems have been considered. Also policy based access control has been determined that each entity is able to specify and process policy by help of *Access Controller* or by itself for security and privacy. In future, the specifications of policies are planned to be fully constructed in declarative manner and the ACL based messages to be considered in a semantic manner. With the help of policy based semantic language, the distributed policy management could be supplied better by inter-platform communication by the platforms that use same ontologies.

Future work will be based on the new version of SEAGENT's role based agent mechanisms. Role based agents would have their own role based access policies for *Access Controller*. Then these agents could have been prioritized according to their goals in the platform [12].

References

[1] Tomás Vlcek and Jan Zach, "Secure fipa compliant agent architecture draft," in *HoloMAS*, 2003, pp. 167–178.

[2] William Stallings, *Cryptography and Network Security Principles and Practices, Fourth Edition*, Prentice Hall, 2005.

[3] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad, *Security Patterns: Integrating Security and Systems Engineering*, John Wiley and Sons Ltd, 2006.

[4] Oguz Dikenelli, Riza Cenk Erdur, Ozgur Gumus, Erdem Eser Ekinci, Onder Gurcan, Geylani Kardas, Inanc Seylan, and Ali Murat Tiryaki, "Seagent: A platform for developing semantic web based multi agent systems. autonomous agents and multi-agent systems," in *AAMAS*, 2005, pp. 1271–1272.

[5] Haralambos Mouratidis, Michael Weiss, and Paolo Giorgini, "Modeling secure systems using an agent-oriented approach and security patterns," *International Journal of Software Engineering and Knowledge Engineering*, vol. 16, no. 3, pp. 471, 2006.

[6] Haralambos Mouratidis, Paolo Giorgini, and Gordon Manson, "When security meets software engineering: a case of modelling secure information systems," *Inf. Syst.*, vol. 30, no. 8, pp. 609–629, 2005.

[7] Haralambos Mouratidis, Michael Weiss, and Paolo Giorgini, "Security patterns meet agent oriented software engineering: A complementary solution for developing secure information systems," in *ER*, 2005, pp. 225–240.

[8] Petr Novák, Milan Rollo, Jirí Hodík, and Tomás Vlcek, "Communication security in multi-agent systems," in *CEEMAS*, 2003, pp. 454–463.

[9] Stefan Poslad and Monique Calisti, "Towards improved trust and security in fipa agent platforms," in *3rd Workshop on Deception, Fraud and Trust In Agent Societies*, 2000.

[10] Lalana Kagal, Tim Finin, and Anupam Joshi, "A policy-based approach to security for the semantic web," in *Proc. 2nd Int'l Semantic Web Conf. (ISWC 2003)*. 2003, p. 402418, Springer-Verlag.

[11] Lalana Kagal, Tim Finin, and Anupam Joshi, "Developing secure agent systems using delegation based trust management," in *In Security of Mobile MultiAgent Systems (SEMAS 02) held at Autonomous Agents and MultiAgent Systems (AAMAS 02)*, 2002.

[12] Salvatore Vitabile, Giovanni Milici, S. Scolaro, Filippo Sorbello, and Giovanni Pilato, "A mas security framework implementing reputation based policies and owners access control," Los Alamitos, CA, USA, 2006, vol. 2, pp. 746–752, IEEE Computer Society.

Documenting and Modeling Multi-agent Systems Product Lines

Ingrid Nunes[1] Uirá Kulesza[2,3] Camila Nunes[1] Carlos J. P. de Lucena[1]

[1] *PUC-Rio, Computer Science Department, LES - Rio de Janeiro - Brazil*
{*ioliveira,camilan,lucena*}*@inf.puc-rio.br*
[2] *Recife Center for Advanced Studies and Systems - Recife - Brazil*
uira@cesar.org.br
[3] *New University of Lisbon - Lisboa - Portugal*

Abstract

In this paper, we explore the use of existing software product line (SPL) approaches to document and model a multi-agent system product line (MAS-PL). Our analysis focuses specifically in the domain analysis and design stages of SPL development. The main aim of our study is to investigate the benefits, limitations and challenges of current SPL and MAS-PL approaches/methodologies to document and model MAS-PL features. Our investigation is illustrated and validated through the use of a web-based conference management system. As a result of our study, we propose the adaptation and extension of existing approaches to address the modeling of MAS-PL features.

1. Introduction

Nowadays, a common scenario in organizations is to develop similar products and to provide different customizations of these products to individual customers. This is typically addressed in an empirical way. Software product lines (SPLs) [18, 3] represent a new trend of software reuse that investigates methods and techniques to build and customize families of applications through a systematic method. Clements & Northrop [3] define a software product line (SPL) as "a set of software intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way ". According to [4], a feature is a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among products in a product line. The main aim of SPL engineering is to analyze the common and variable features of applications from a specific domain, and to develop a reusable infrastructure that supports the software development. This set of applications is called a family of products.

Over the past few years, several methods have been published to address the problems and challenges of SPL engineering [12, 18, 8]. Some of them only propose methodological guidelines, not specifying how to design or implement the SPL, meaning developers have to create their own way to develop the product line. Some of these methodologies propose a complete SPL development process based on existing paradigms, such as component-based [1] or object-oriented [8] software development. However, there are new trends, such as multi-agent systems (MASs) [10, 19], which are not considered by the current SPL methodologies. MASs have emerged as a new software paradigm to help in the development of complex software systems, which contain properties such as autonomy, reactivity, proactiveness and social ability. Recently, new approaches [16, 6] were proposed designed to explore the benefits of integrating SPL and Agent-Oriented Software Engineering (AOSE) techniques. Nevertheless, there are still many challenges to overcome in the development of multi-agent systems product lines (MAS-PLs) [17].

In this context, this paper investigates the adoption of proposed SPL and MAS-PL methodologies in the documentation and modeling of MAS-PL. During this process, we had to deal with challenges, such as how the features can be documented, modeled and modularized throughout the entire domain engineering process. A product line of conference management systems that includes the implementation of several optional agency features is used to illustrate and validate our study. Some adaptations and extensions of current SPL approaches are also proposed in order to address their identified deficiencies in the modeling and documentation of more complex agency features.

The remainder of this paper is organized as follows. Section 2 presents some existing SPL methodologies. In Section 3, an overview of the ExpertCommittee case study is

presented, giving some details about its development. In Section 4, we show how we have modeled and documented our product line at the domain analysis and domain design phases. We present some discussions in the Section 5. Finally, the conclusions and directions for future works are discussed in Section 6.

2. Background

We have studied and compared some existing SPL methodologies. Almost all methods focus on the description of SPL properties at a very high level of abstraction and give no guidance on how the required flexibility should be realized at the implementation level. In our initial comparative study [14], we used the same evaluation framework of [13] to compare the SPL and MAS-PL methodologies. The goal of this evaluation was to obtain an overview of the methodologies and not necessarily to rate them. Subsequently, we analyzed how the investigated approaches could deal with the documentation and modeling of agency features. Table 1 presents partial results of this analysis. Due to space restrictions in this paper, we only reported the results of this second part of our study. For additional details of our study, please refer to [14].

Table 1. Methodologies Comparison.

Methodology	Domain Analysis	Domain Design
FORM	Feature diagram with composition rules	Subsystem, Process Model and Module Models
Framework [18]	Reusable, textual and model-based requirements, variability model	Reference architecture, refined variability model, mapping from design artifacts to requirements artifacts
PLUS	Requirements model consisting of a use case model and feature model	Static and dynamic models, feature/class dependencies, design of component-based software architecture
MacMAS extension [16]	Feature Model (features are goals)	Acquaintance Organization, Traceability and Role Models
Approach [6]	Role Schema, Role Variation Point	-

Commonly the SPL approaches adopt feature models as the typical notations to specify the SPL features. FORM [12] provides a feature modeling method for analyzing and capturing the common and variable features of SPLs and their respective interdependencies. The features are organized into a coherent model referred to as a feature model, which models the features of a product line as a tree, indicating mandatory, optional and alternative features. Features are essential abstractions that both customers and developers understand. Pena et. al. [16] also proposes the use of feature models, but the features are the goals of the agents. Goals are not a detail of the system that is visible to the end user; therefore, they should not appear in a feature model. [18] document variabilities through a variability model, which models what varies from one product to another with the explicit indication of the variation points and variants. Furthermore, it also motivates the definition of explicit tracing links between the variations points/variants from the variability model and other analysis and design models (e.g., use cases and class diagrams). PLUS [8] proposes a feature model based on UML notation, but contains the same information of traditional feature models. Almost all the approaches do not address explicitly the modeling of the SPL requirements. The PLUS approach defines a customization of the use case model to specify and document the SPL requirements. Dehlinger & Lutz [6] adopt a product-line-like view of an agent-based software system and proposes a requirements specification template to capture and reuse dynamically changing configurations of agents for future similar systems.

In the domain design, most of the SPL approaches investigated only provide support to document and detail the SPL architectures in a very high-level manner. FORM proposes the modeling of an SPL architecture using three models: (i) *subsystem model* - presents the overall system structure; (ii) *process model* - details the dynamic behavior of the system; and (iii) *module model* - specifies each reusable component of the architecture. PLUS adopts traditional UML models marked with additional stereotypes to classify the system classes. It mentions the use of agents in the design of an SPL architecture, but it does not define a way to document it. Table 1 shows that the other investigated approaches [18, 6] do not provide explicit support to specify and model the SPL architecture and its respective components.

3. ExpertCommittee Case Study Overview

Our approach was developed based on our experience with the ExpertCommittee (EC) [15] case study, a multi-agent system product line for the web domain.

The EC is a conference management system, developed as a typical web-based application whose aim is to manage the paper submission and reviewing processes from conferences and workshops. The EC system provides functionalities to support the complete process of conference management. Each of these functionalities can be executed by an appropriate user type of the system, such as conference chair, program committee members, authors and reviewers.

This MAS-PL was developed in an evolutionary way. We present details about the MAS-PL development (Section 3.1). After that, we discuss some MAS particular variability types that we have identified in our case study (Section 3.2).

3.1. The EC MAS-PL

We developed our case study considering that an evolving system can be seen as an SPL, because the features

that are common to all versions of the system comprise the core architecture of the product line. Thus, each version of the system, which has new features, characterizes a new product.

Our MAS-PL was developed in an evolutionary way. There were three versions of the EC. The first version of the EC is a typical web-based application composed of the mandatory features that support the process of conference management. It was structured according to the Layer architectural pattern [7]. The second version of the EC system contains features that are related to autonomous behavior, such as deadline and pending tasks monitoring, and it has also some new features that add new functionalities to the system as well. The software agent abstraction was used to model and implement the autonomous behavior added to the original EC system.

The third and last version of the EC system was implemented by applying a series of refactorings in version 2. The system was restructured to make the (un) plugging of optional features possible. Each optional feature was modularized by using a combination of OO design patterns and techniques with Spring[1] configuration files that allows the injecting of dependencies inside the variable points of the EC SPL architecture, which can be seen in Figure 1.

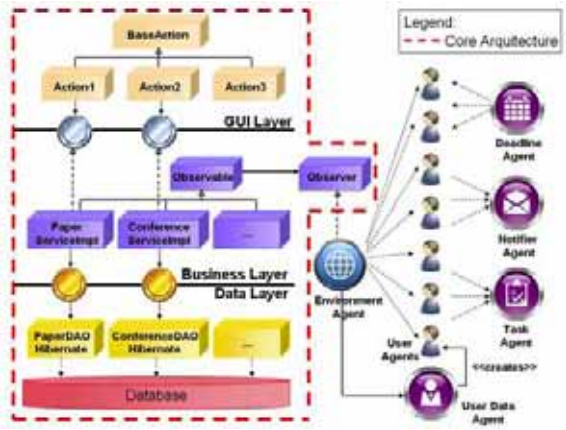


Figure 1. EC MAS-PL Architecture.

3.2. Dealing with Variability

Different kinds of variability were identified in the EC MAS-PL. In our case study, we mainly explored the variabilities related to autonomous behavior and their respective implementation using software agents. Throughout this paper, these kinds of features are called agency feature. Next we briefly describe them:

[1] http://www.springframework.org/

New Autonomous Behavior. We had to introduce agents into the architecture when we added autonomous behavior to the system. The *Task Management* feature implied the addition of a new agent in the system, which can be present or not, depending on the product being derived;

New Behavior for an Agent or Role. Some features have an impact inside the agent or the role. They allow defining agent internal variabilities by defining specific new behaviors of agents. The *Conference Suggestion Feature* is an autonomous feature; thus, the user agent, or more specifically the author role, performs it. When a paper is registered in a conference, the author role perceives it and sends suggestions of related conferences for the author who has registered his/her paper;

New Role for an Agent. Each role of the EC has a corresponding role in the user agent when a product has some autonomous behavior. However, not all roles are mandatory, such as the role *Reviewer*. Thus, roles must be modeled in a way that they can be (un) plugged.

Almost all the autonomous behavior features are accomplished by the collaboration of different agents. In our study, we have identified that many of these features are typically addressed by a different set of components and agents from the SPL architecture. In this way, a particular challenge of our study was to document and model the structure and behavior of these crosscutting features in domain analysis and design.

4. Modeling and Documenting Agency Features

In this section, we discuss the modeling and documentation of the agency features from the EC MAS-PL, presented in Section 3. We focus specifically on the domain analysis and design stages. We have initially analyzed how existing SPL and MAS-PL approaches can deal with the specification and modeling of agency features. Based on the deficiencies and lack of expressivity of these existing approaches, we propose new extensions to document the agency features of the EC MAS-PL. The main aim of our work is to define a set of guidelines to model and document agency features along all SPL development stages.

4.1. Domain Analysis

The domain analysis stage defines activities for eliciting and documenting the common and variable requirements of an SPL. It is concerned with the definition of the domain and scope of the SPL, and specifies the common and variable features of the SPL to be developed. In our study, we

have analyzed how the modeling and documentation notations of current SPL approaches can deal with agency features. Table 1 shows the results obtained considering the SPL methodologies investigated in our study.

The EC MAS-PL features modeling and documentation was supported by the feature model proposed in [5]. It is an evolution of the original feature model proposed in [11] and also adopted by FORM. Figure 2 shows a partial view of this feature model. The features that were in all the versions are the mandatory ones. Features that made part of only some versions or varied from one version to another one are the optional features.

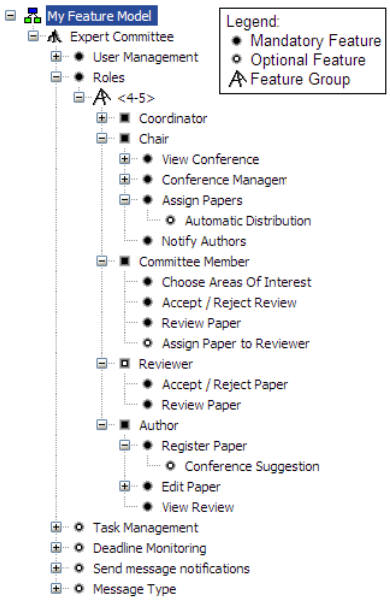


Figure 2. Feature Model.

The way proposed in the PLUS method was quite adequate to model our use cases. Use cases are grouped in packages according to the feature to which it was related. In this approach, stereotypes are used to indicate if a use case is mandatory (kernel), alternative or optional. The method also proposes a feature dependency table to map use cases to each feature. We adopted these tables instead of the graphical notation of [18]. Figure 3 shows a partial view of the EC MAS-PL use case model. It contains three kernel use cases, one optional use case related to the reviewer role and two agency features: task management and conference suggestion. The following adaptations were applied to the use case notation proposed in [8] to better specify the agency features: (i) agents were represented with the same symbol as actors and are associated to the use cases with which they are involved; (ii) the *<<agency feature>>* stereotype was adopted to indicate that the use cases of a specific package is related to an agency feature.

The detailed description of the EC MAS-PL use cases

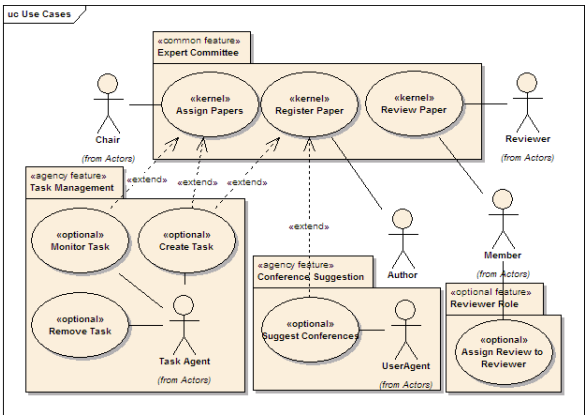


Figure 3. Use Case Diagram.

was carried out in the following way: (i) the kernel use cases were described using the common documentation provided by existing UML methods; and (ii) the agency features were documented using the template depicted in Table 2. This new template details important information to understand the interactions between the agency feature and other ones, such as: the event that starts the use case, the agents and roles that are involved and if the feature is mandatory, optional or alternative. We did not used the template proposed in [6] because it is a too low-level specification and it addresses the internal variability of the agents.

Table 2. Agency Feature Description.

Agency Feature: Conference Suggestion
Reuse Category: Optional
Dependency: Extends *Register Paper* Use Case
Description: When a paper is registered to a conference
Event: paper was registered to a conference
Agent/Roles: user agent / author role, notifier agent
Main Flow: 1. User registers a paper to a conference. 2. User Agent perceives the change in the environment. 3. Author role detects the conferences that have areas of interest similar to the ones of the registered paper and creates a message to be sent to the user. 4. Author role sends a message to the Notifier Agent requesting to send the message to the user. 5. Notifier Agent sends the message.

4.2. Domain Design

The domain design aims at defining an architecture that addresses both the common and variable features of an SPL. A set of components and core assets can be specified as part of the SPL architecture. The modularization of features must also be taken into account during the design of the architecture core assets to allow the (un) plugging of features.

The EC MAS-PL architecture was documented in our case study in two different levels: (i) a component view - that illustrates the main components (or subsystems) of the

SPL architecture; and (ii) a logical view - that details the different components defined for the SPL architecture in terms of UML class diagrams. Figures 1 and 4 show, respectively, the component and logical view of the EC architecture. The component view details the web system layers and the deployed agents that execute inside this system. The component view gives not only an overall overview of the SPL architecture components and agents, but also expresses their organization in runtime.

The logical view details the architecture components and agents in terms of UML class diagrams. Similar to PLUS, we used stereotypes to classify the classes, but our classification was mandatory (kernel), optional or alternative. The classes of different components can be organized in packages, or they also can be colored to characterize a specific component. Figure 1 shows the main components of the EC MAS-PL (GUI, Business and Data Layers), and the different agents responsible for implementing the autonomous behavior of the system. Each different agency feature of the MAS-PL can be detailed using: (i) a separate class diagram that only contains the classes responsible for implementing that feature and alternatively the classes that are related with it; (ii) a colored indication in the main class diagram that shows the elements (classes, interfaces, methods) related to the implementation of that feature. It is exemplified in Figure 4; and (iii) a specific design template that details the components and agents involved in the realization of an agency feature, and their respective interactions.

Table 3 shows the design template of the Conference Suggestion agency feature. It details the goals, entities, events and execution plan related to the conference suggestion feature provided by a set of agents. It complements the agency feature description provided in domain analysis (Figure 2) by detailing the communication of the different system agents and the environment. While the class diagrams of an agency feature describe the elements that modularize it, our template design details the dynamics of the agents involved in its realization.

Table 3. Agency Feature Design Description.

Agency Feature: Conference Suggestion	
Goal: Send conference suggestions to users	
Entities: EnvironmentAgent, UserAgent, NotifierAgent, AuthorRole and ConferenceService.	
Events Generated: SendMessage **Events Perceived:** RegisterPaper	
Plan:	
Environment Agent	*Action:* send message to User Agents *Message Content:* paper registered
User Agent	*Action:* creates Author Role and adds it to the agent *Condition:* user is the first author of the paper
Author Role	*Action:* send message to Conference Service *Message Content:* conferences related to the conference the author has registered
Conference Service	*Action:* send message to Author Role *Message Content:* related conferences
Author Role	*Action:* creates user message with conferences returned *Action:* send message to Notifier Agent *Message Content:* user message to be sent to the user
Notifier Agent	*Action:* send user message

5. Discussions

In this section, we discuss some lessons learned and challenges that we have found when documenting the agency features of EC MAS-PL. These lessons learned offer directions for a methodology for developing MAS-PL that we are currently defining.

Agency Feature Documentation using SPL methodologies. During the modeling and documentation of the EC MAS-PL, we have identified that most of the SPL methodologies provide useful notations to model the agency features. However, none of them completely covers their specification. Agent technology provides particular characteristics that need to be considered in order to take advantage of this paradigm. In our case study, we adopted a different strategy to model the SPL agency features. We started modeling the agency features using only the notations provided by SPL methodologies to investigate their expressivity. After that, we adapted and complemented the selected notations to improve the documentation of the agency features. The domain analysis and design templates were created in this context.

MAS-PL methodologies. The investigated MAS-PL methodologies do not address development scenarios of traditional SPL architectures using agent technology. Instead, they adopt an existing MAS methodology as a base and extend it with SPL techniques for a particular purpose. Pena et. al. [16] adapt the Methodology for analyzing Complex MultiAgent Systems (MaCMAS) to deal with evolving systems. Dehlinger & Lutz [6] have proposed an extensible agent-oriented requirements specification template for distributed systems that supports safe reuse. Their proposal adopts a product line to promote reuse in MASs, which was developed using the MaCMAS and the Gaia methodologies. The main problems that we have observed when using these MAS-PL methodologies to model and document the EC MAS-PL were: (i) they do not offer a complete solution to address the modeling of agency features in both domain analysis and design; and (ii) they suggest the introduction of complex and heavyweight notations that are difficult to understand when adopted in combination with existing notations (e.g. UML) and do not capture explicitly the separated modeling of agency features.

Crosscutting agency features. Many of the agency features are implemented by a set of different system components, agents and classes. They are characterized as crosscutting features, because their design and implementation are typically spread and tangled along different system modules. In our study, we observed that the current SPL methodologies do not provide clear support to deal with the

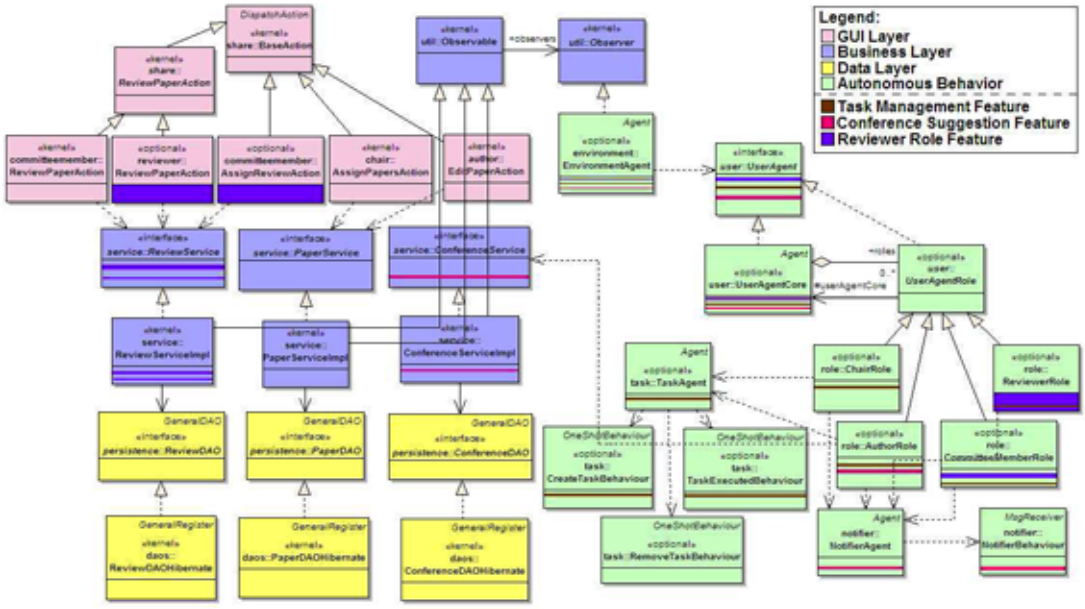


Figure 4. Class Diagram of the EC product line.

documentation of these crosscutting features. In domain design, we have proposed a template design to help the documentation of the agency features. It allows specifying how the different design elements interact to address a specific agency feature. We are currently investigating how existing aspect-oriented approaches [9, 2] can help the visual documentation of the agency features in combination with our templates.

6. Conclusions and Future Work

In this paper, we presented an exploratory study that analyzed and discussed how existing SPL approaches can help the documenting and modeling of multi-agent system product lines (MAS-PLs). Different agency features were presented, which were added to an existing web-based conference management system as optional features. Three types of agency variabilities were addressed in our paper: addition of agents; addition of plans; and addition of roles. Most of the MAS-PL documentation was supported by the PLUS approach, showing the effectiveness of current SPL approaches to document MAS-PL. However, the documentation of the agency features required the creation of additional templates to specify: (i) the interdependencies and relationships between core functionalities (mandatory use cases) and optional agency features (optional use cases) in domain analysis; and (ii) the elements and dynamics responsible to address a given agency feature in domain design.

We are currently working on the development of a methodology that allows an explicit documentation and tracing of agency features throughout the SPL development process. The proposed methodology aims to be simple and systematic. We believe that due to the high complexity of many SPL methodologies, many of them are not used in practice. Different and new abstractions have been proposed in these methodologies, making the understanding and adoption of them difficult. Our methodology is being organized as a process framework composed of: (i) a core - that defines a set of mandatory activities and artifacts; and (ii) specific customizations - that specify additional activities and artifacts to the core according to specific scenarios that need to be addressed. Our approach aims to be systematic in the sense of providing clear and detailed guidelines about how developers should use it.

References

[1] C. Atkinson, J. Bayer, and D. Muthig. Component-based product line development: The KobrA approach. In P. Donohoe, editor, *SPLC*, pages 289–309, 2000.

[2] S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach (The Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, March 2005.

[3] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA, USA, 2002.

[4] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

[5] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Staged configuration using feature models. In *SPLC*, pages 266–283, 2004.

[6] J. Dehlinger and R. R. Lutz. A Product-Line Requirements Approach to Safe Reuse in Multi-Agent Systems. In *SELMAS*, pages 1–7, New York, NY, USA, 2005. ACM Press.

[7] M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, November 2002.

[8] H. Gomaa. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[9] I. Jacobson and P.-W. Ng. *Aspect-Oriented Software Development with Use Cases (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2004.

[10] N. R. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.

[11] K. Kang, S. Cohen, J. Hess, W. Novak, and Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie-Mellon University, November 1990.

[12] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, 1998.

[13] M. Matinlassi. Comparison of software product line architecture design methods: Copa, fast, form, kobra and qada. In *ICSE*, pages 127–136, Washington, DC, USA, 2004. IEEE Computer Society.

[14] I. Nunes. Towards a multi-agent product line development methodology, 2008. http://www.inf.puc-rio.br/ ioliveira/maspl/.

[15] I. Nunes, C. Nunes, U. Kulesza, and C. Lucena. Developing and evolving a multi-agent system product line: An exploratory study. In *AOSE (to appear)*, 2008.

[16] J. Pena, M. G. Hinchey, M. Resinas, R. Sterritt, and J. L. Rash. Designing and managing evolving systems using a MAS product line approach. *Science of Computer Programming*, 66(1):71–86, 2007.

[17] J. Pena, M. G. Hinchey, and A. Ruiz-Cortés. Multi-agent system product lines: challenges and benefits. *Commun. ACM*, 49(12):82–84, 2006.

[18] K. Pohl, G. Bckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, New York,USA, 2005.

[19] M. Wooldridge and P. Ciancarini. Agent-Oriented Software Engineering: The State of the Art. In P. Ciancarini and M. Wooldridge, editors, *AOSE*, volume 1957, pages 1–28. Springer-Verlag, Berlin, 2000.

A Study of the Model Explosion Problem in CTL Model Update

Yulin Ding
School of Computer Science
The University of Adelaide
Adelaide, SA 5005 Australia
Email: yulin@cs.adelaide.edu.au

Yan Zhang
School of Computing & Mathematics
University of Western Sydney
Penrith South DC, NSW 1797 Australia
Email: yan@scm.uws.edu.au

***Abstract*— Computation Tree Logic (CTL) model update is an approach for software verification and modification, where minimal change is employed to generate admissible models that represent the corrected software design. In this paper, we first apply CTL model update to a model based on the well known Andrew File System protocol, and demonstrate the process of discovering an admissible model explosion problem. We then propose a new update principle named minimal change with maximal reachable states to solve this problem. Our experimental results show that in the case of updating the Andrew File System protocol model, the new CTL update approach significantly narrows down the admissible models to fewer committed models. We provide a thorough study on the semantics and complexity properties on optimizing CTL model update.**

I. Introduction

As a promising formal method, automated verification has played an important role in computer science development. The SMV model checker [2] was first successfully applied to software protocols, the Andrew File Systems in 1995 by [9], which was a milestone showing that the SMV model checker applied to not only hardware verification but also to software verification. After the first successful SMV compiler, the enhanced model checking compilers, NuSMV [3] and Cadence SMV [7], were developed. During the model checking, there was a state explosion problem, which significantly increased the SMV model checking search space. The introduction of OBDD in SMV model checker eventually handles the state explosion problem in a significant way [2], [6].

Along with the development of model checking, error repairing has also developed using a formal methods approach. For example, Buccafurri and his colleagues [1] applied AI techniques to model checking and error repairing. We have recently developed a formal approach for CTL model update to attempt a new method of software error repairing [4]. We have implemented a prototype of CTL model update and applied this system to 3 well known model checking examples [5].

In this paper, we provide an investigation, both on experimental and theoretical aspects, on a model explosion problem associated with the CTL model update to optimize the previous update results. In section II, we first present an overview of CTL model update. In section III, we address the problem of admissible model explosion through the update process of the Andrew File System 1 model. In section IV, we propose a new CTL model update principle called minimal change with maximal reachable states to deal with the model explosion problem. In section V, we provide some semantic characterizations for the new CTL model update process. In section VI, the associated complexity results of the characterizations in the previous section are derived. Finally in section VII , we conclude the paper with some discussions and future work.

II. CTL Model Update: An Overview

A. CTL Syntax and Semantics

Definition 1: [2] Let AP be a set of atomic propositions. A Kripke model M over AP is a three tuple $M = (S, R, L)$ where 1. S is a finite set of states, 2. $R \subseteq S \times S$ is a transition relation, 3. $L : S \to 2^{AP}$ is a function that assigns each state with a set of atomic propositions.

Definition 2: [6] Computation tree logic (CTL) has the following syntax given in Backus naur form:

$\phi ::= \top | \perp |p|(\neg\phi)|(\phi_1 \wedge \phi_2)|(\phi_1 \vee \phi_2)|\phi_1 \to \phi_2|AX\phi$
$|EX\phi|AG\phi|EG\phi|AF\phi|EF\phi|A[\phi_1 \cup \phi_2]|E[\phi_1 \cup \phi_2]$

where p is any propositional atom.

A CTL formula ϕ is evaluated on a Kripke model M and satisfiable. A path in M from a state s is an infinite sequence of states $\pi \stackrel{def}{=} [s_0, s_1, \cdots, s_{i-1}, s_i, s_{i+1}, \cdots, s_j, \cdots]$ such that $s_0 = s$, $(s_i, s_{i+1}) \in R$ holds for all $i \geq 0$, $(s_i, s_{i+1}) \subseteq \pi$ and $s_i \in \pi$. s_i is a state *earlier* than s_j in π as $s_i < s_j$. A state s is called true state for ϕ if $s \models \phi$ and called false state for ϕ if $s \not\models \phi$.

Definition 3: [6] Let $M = (S, R, L)$ be a Kripke model for CTL. Given any s in S, we define whether a CTL formula ϕ holds in state s. We denote this by $(M, s) \models \phi$. The satisfaction relation $\models$ is defined by structural induction on CTL formulas:

1) $(M, s) \models \top$ and $(M, s) \not\models \perp$ for all $s \in S$.
2) $(M, s) \models p$ iff $p \in L(s)$.
3) $(M, s) \models \neg\phi$ iff $(M, s) \not\models \phi$.
4) $(M, s) \models \phi_1 \wedge \phi_2$ iff $(M, s) \models \phi_1$ and $(M, s) \models \phi_2$.
5) $(M, s) \models \phi_1 \vee \phi_2$ iff $(M, s) \models \phi_1$ or $(M, s) \models \phi_2$.
6) $(M, s) \models \phi_1 \to \phi_2$ iff $(M, s) \not\models \phi_1$, or $(M, s) \models \phi_2$.
7) $(M, s) \models AX\phi$ iff for all s_1 such that $(s, s_1) \in R$, $(M, s_1) \models \phi$.
8) $(M, s) \models EX\phi$ iff for some s_1 such that $(s, s_1) \in R$, $(M, s_1) \models \phi$.

9) $(M, s) \models AG\phi$ holds iff for all paths $[s_0, s_1, s_2, \cdots]$, where $s_0 = s$, and all s_i along the path, $(M, s_i) \models \phi$.
10) $(M, s) \models EG\phi$ holds iff there is a path $[s_0, s_1, s_2, \cdots]$, where $s_0 = s$, and for all s_i along the path, $(M, s_i) \models \phi$.
11) $(M, s) \models AF\phi$ holds iff for all paths $[s_0, s_1, s_2, \cdots]$, where $s_0 = s$, there is some s_i in the path such that $(M, s_i) \models \phi$.
12) $(M, s) \models EF\phi$ holds iff there is a path $[s_0, s_1, s_2, \cdots]$, where $s_0 = s$, and for some s_i along the path, $(M, s_i) \models \phi$.

B. Minimal Change for CTL Model Update

Definition 4: [4], [5] (**CTL Model Update**) Given a CTL Kripke model $M = (S, R, L)$ and a CTL formula ϕ such that $\mathcal{M} = (M, s_0)$ and $\mathcal{M} \not\models \phi$, where $s_0 \in S$, an updated model derived from $\mathcal{M}$ to satisfy ϕ, is a new CTL Kripke model $M' = (S', R', L')$ such that $\mathcal{M}' = (M', s_0') \models \phi$ where $s_0' \in S'$. We use $Update(\mathcal{M}, \phi)$ to denote the result $\mathcal{M}'$.

Update is achieved by applying a combination of primitive update operations PU1, PU2, PU3, PU4 and PU5 [4], [5]. These primitive updates are defined as follows:

PU1: Adding a relation

Given $M = (S, R, L)$, its updated model $M' = (S', R', L')$ is the result of M having only added one new relation. That is $S' = S$, $L' = L$, and $R' = R \cup \{(s_{ar}, s_{ar2})\}$ where $(s_{ar}, s_{ar2}) \notin R$ for one pair of $s_{ar}, s_{ar2} \in S$.

PU2: Removing a relation

Given $M = (S, R, L)$, its updated model $M' = (S', R', L')$ is the result of M having only removed one existing relation. That is, $S' = S$; $L' = L$, and $R' = R - \{(s_{rr}, s_{rr2})\}$ where $(s_{rr}, s_{rr2}) \in R$ for one pair of $s_{rr}, s_{rr2} \in S$.

PU3: Substituting a state and its associated relation(s)

Given $M = (S, R, L)$, its updated model $M' = (S', R', L')$ is the result of M having only substituted one existing state and its associated relation(s). That is, $S' = S[s/s_{ss}]$ (i.e. S' are the set of states where one state s in S is substituted by s_{ss}), $R' = R \cup \{(s_i, s_{ss}), (s_{ss}, s_j) \mid (s_i, s), (s, s_j) \in R\} - \{(s_i, s), (s, s_j) \mid (s_i, s), (s, s_j) \in R\}$, and for all $s \in S \cap S'$, $L'(s) = L(s)$, and $L'(s_{ss})$ is a set of true variables assigned in s_{ss}.

PU4: Adding a state and its associated relation(s)

Given $M = (S, R, L)$, its updated model $M' = (S', R', L')$ is the result of M having only added one new state and its associated relation(s). That is, $S' = S \cup \{s_{as}\}$, $R' = R \cup \{(s_i, s_{as}), (s_{as}, s_j) \mid$ for some $s_i, s_j \in S'\}$, and for all $s \in S \cap S'$, $L'(s) = L(s)$, and $L'(s_{as})$ is a set of true variables assigned in s_{as}.

PU5: Removing a state and its associated relation(s)

Given $M = (S, R, L)$, its updated model $M' = (S', R', L')$ is the result of M having only removed one existing state and its associated relation(s). That is, $S' = S - \{s_{rs} \mid s_{rs} \in S\}$, $R' = R - \{(s_i, s_{rs}), (s_{rs}, s_j) \mid$ for some $s_i, s_j \in S\}$, and $L'(s) = L(s)$ for all $s \in S \cap S'$.

Based on these primitive update operations, we can define a minimal principle for CTL model update. Given any two sets X and Y, the *symmetric difference* between X and Y is denoted as $Diff(X, Y) = (X - Y) \cup (Y - X)$. Given two CTL models $M = (S, R, L)$ and $M' = (S', R', L')$, for each primitive operation PUi $(i = 1, \cdots, 5)$, $Diff_{PUi}(M, M')$ denotes the differences between two CTL models where M' is a resulting model from M, which infers that several operations of this type may occur. Since PU1 and PU2 only change relations, we define $Diff_{PUi}(M, M') = Diff(R, R')$ $(i = 1, 2)$. For operations PU3, PU4 and PU5, on the other hand, we define $Diff_{PUi}(M, M') = Diff(S, S')$ $(i = 3, 4, 5)$. Then we specify $Diff(M, M') = (Diff_{PU1}(M, M'), \cdots, Diff_{PU5}(M, M'))$.

Let M, M_1 and M_2 be three CTL models. We denote $Diff(M, M_1) \preceq Diff(M, M_2)$ iff (1) for each i $(i = 1, \cdots, 5)$, $Diff_{PUi}(M, M_1) \subseteq Diff_{PUi}(M, M_2)$; or (2) $Diff_{PUi}(M, M_1) \subseteq Diff_{PUi}(M, M_2)$ for $i = 1, 2, 4, 5$, and $|Diff_{PU3}(M, M_1)| = |Diff_{PU3}(M, M_2)|$ implies $Diff(s, s_1) \subseteq Diff(s, s_2)$, if any state s in M is substituted by s_1 in M_1 or s_2 in M_2 respectively.

Definition 5: [4], [5] (**Closeness Ordering**) Given three CTL Kripke models M, M_1 and M_2, where M_1 and M_2 are obtained from M by applying PU1-PU5 operations, we say that M_1 is *closer* or *as close to* M *as* M_2, denoted as $M_1 \leq_M M_2$, iff $Diff(M, M_1) \preceq Diff(M, M_2)$. We denote $M_1 <_M M_2$ if $M_1 \leq_M M_2$ and $M_2 \not\leq_M M_1$.

Definition 6: [4], [5] (**Admissible Update**) Given a CTL Kripke model $M = (S, R, L)$, $\mathcal{M} = (M, s_0)$ where $s_0 \in S$, and a CTL formula ϕ, $Update(\mathcal{M}, \phi)$ is called *admissible* if the following conditions hold: (1) $Update(\mathcal{M}, \phi) = (M', s_0') \models \phi$ where $M' = (S', R', L')$ and $s_0' \in S'$; and (2) there does not exist another resulting model $M'' = (S'', R'', L'')$ and $s_0'' \in S''$ such that $(M'', s_0'') \models \phi$ and $M'' <_M M'$.

Theorem 1: [4], [5] Let $M = (S, R, L)$ be a Kripke model and $\mathcal{M} = (M, s_0)$ and $\mathcal{M} \not\models AG\phi$, where $s_0 \in S$ and ϕ is a propositional formula. Then an admissible updated model $\mathcal{M}' = Update(\mathcal{M}, AG\phi)$ can be obtained by the following: for each path starting from s_0: $\pi = [s_0, \cdots, s_i, \cdots]$:

1) if for all $s < s_i$ in π, $s \models \phi$ but $s_i \not\models \phi$, PU2 is applied to s_i to remove relation (s_{i-1}, s_i), or PU5 is applied to s_i to remove s_i and its associated relations, or
2) PU3 is applied to all states s in π not satisfying ϕ to substitute s with $s^* \models \phi$ and $Diff(s, s^*)$ to be minimal.

III. A Model Explosion Problem

A. The Scenario of AFS1

AFS1 is abbreviation of the Andrew File System 1 [9]. It is a cache coherence protocol for a distributed file system. A client has two initial states: either it has no files or it has one or more files but no beliefs about their validity. If the protocol starts with the client having suspect files, then the client may request a validation for a file from the server. If the file is invalid then the client requests a new copy and the run terminates. If the file is valid, the protocol simply terminates. AFS1 is abstracted as a model with one client, one server and one file. Fig. 1 shows the state transition diagrams with single client and server modules. The nodes are labelled with the

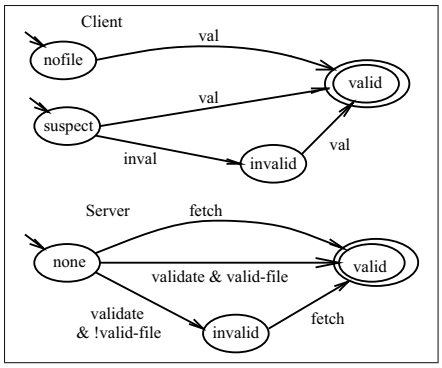


Fig. 1. State Transition Diagrams for AFS1

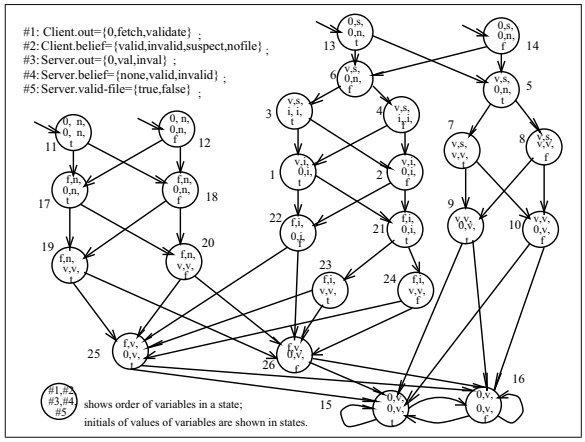


Fig. 2. The CTL Kripke structure of AFS1

value for the state variable, $belief$; the arcs, by the name of the message received that causes the state transition. A run of the protocol begins at an initial state (one of the leftmost nodes) and ends in a final state (one of the rightmost nodes).

The client's belief about a file has 4 possible values $\{nofile, valid, invalid, suspect\}$, where *nofile* means that the client cache is empty; $valid$, if the client believes its cached file is valid; $invalid$ if it believes its caches file is not valid; $suspect$, if it has no belief about the validity of the file (it could be $valid$ or $invalid$). The server's belief about the file cached by the client ranges over $\{valid, invalid, none\}$, where $valid$, if the server believes that the file cached at the client is valid; $invalid$, if the server believes it is not valid; *none*, if the server has no belief about the existence of the file in the client's cache or its validity.

The set of messages that the client may send to the server is $\{fetch, validate\}$. The message $fetch$ stands for a request for a file. The $validate$ message is used by the client to determine the validity of the file in its cache. The set of messages that the server may send to the client is $\{val, inval\}$. The server sends the $val(inval)$ message to indicate to the client that its cached file is valid (invalid).

The specification property for AFS1 (formula (1)) is:
$AG((Server.belief = valid) \rightarrow (Client.belief = valid))$.
Our model updater will update the AFS1 model to derive admissible models which satisfy the above specification property.

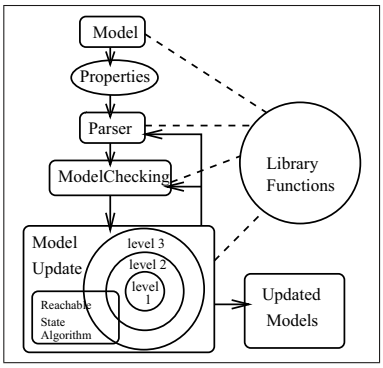


Fig. 3. The flow diagram of the Model Update System

The paper [9] provides SMV model definitions (e.g. AFS1.smv) as input to the SMV model checker. However, the paper does not contain an accurate Kripke model for AFS1. We have used NuSMV [3] to derive the Kripke structure for the loaded model (AFS1) as in [5]. In the AFS1 Kripke model (Fig. 2), there are 26 reachable states out of a total 216 states, and 52 transitions in between the reachable states. The model contains 5 variables and each individual variable has 2 to 4 possible values. The variables are "Client.out", which ranges over $\{0, fetch, validate\}$, "Client.belief" over $\{valid, invalid, suspect, nofile\}$, "Server.out" over $\{0, val, inval\}$, "Server.belief" over $\{none, valid, invalid\}$ and "Server.valid-file" over $\{true, false\}$.

B. Model Updating AFS1

We have developed a prototype of the CTL model updater in Linux C as the implementation of our algorithms. The CTL model updater includes library functions, predefined model definition functions, a specification string parser, model checking functions and model update functions. The diagram of the code structure is shown in Fig. 3. The model updater has been successfully applied to the AFS1 model.

We identify the false states which do not satisfy the specification property for AFS1 (formula (1)) as $\{19, 20, 23, 24, 7, 8\}$.

In AFS1, because each false state is on a different path, PU2, PU3 or PU5 is applied to each false state (i.e. each false path as well) one time to update the model according to Theorem 1. Thus, the combination of total admissible models are $(C_1^3)^6 = 729$. One of the admissible models which cannot retain maximal unchanged reachable states is shown in Fig. 4.

C. Classifying Different Admissible Models by Reachable State Characteristics

There are many more admissible models than we expect. This phenomena is called *admissible model explosion*. We should classify the admissible models and minimize the number of admissible models. We should preserve the maximal unchanged reachable states in an admissible model because this will retain the maximum amount of the original model structure. The *unchanged reachable states* mean that the reachable states in an admissible model are also in the original model.

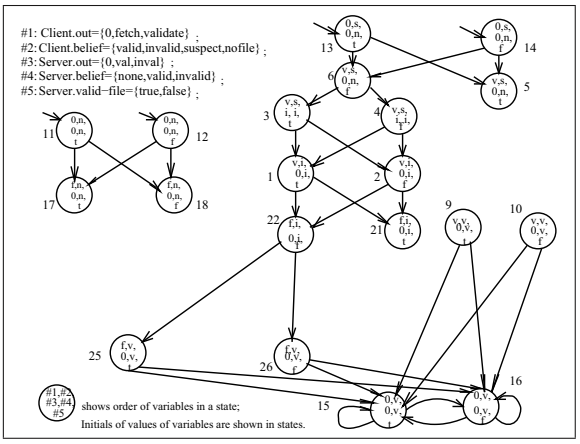


Fig. 4. One of the admissible models from AFS1

To simplify our analysis, we divide the model AFS1 into two self contained sub models. One is the sub model AFS1-1 on the left side of the Kripke model in Fig. 2, which contains states $\{11, 12, 17, 18, 19, 20, 25, 26, 15, 16\}$, where 11 and 12 are initial states. The other sub model is AFS1-2 on the right side and contains states $\{13, 14, 6, 5, 3, 4, 1, 2, 22, 21, 23, 24, 25, 26, 7, 8, 9, 10, 15, 16\}$, where 13 and 14 are initial states.

There are $(C_1^3)^2 = 9$ admissible models of AFS1-1, where $2 \times (C_1^1 \times C_1^3) - (C_1^1)^2 = 2 \times 3 - 1 = 5$ preserve maximum unchanged reachable states. These admissible models are derived from the method where PU3 is applied to at least one of states 19 and 20. $(C_1^2)^2 = 4$ admissible models, which are derived from either PU2 or PU5 applied to states 19 and 20, do not preserve maximum unchanged reachable states. The different sets of reachable states in admissible models are:
updated$_1$ (AFS1-1)= $\{11(or12), 17, 18\}$,
updated$_2$(AFS1-1)= $\{11(or12), 17, 18, 19', 25, 26, 15, 16\}$,
updated$_3$(AFS1-1)= $\{11(or12), 17, 18, 20', 25, 26, 15, 16\}$,
updated$_4$(AFS1-1)= $\{11(or12), 17, 18, 19', 20', 25, 26, 15, 16\}$.
Admissible models with updated$_1$(AFS1-1) are the results from PU2 or PU5 applied to states 19 and 20. Admissible models with updated$_2$(AFS1-1) are the results from 2 combinations of updates: state 19 updated using PU3 to derive a new state $19'$ and state 20 updated using PU2 or PU5. Admissible models with updated$_3$(AFS1-1) are results by applying PU3 to state 20 and PU2 or PU5 to state 19. The admissible model with updated$_4$(AFS1-1) is a result by applying PU3 to states 19 and 20. We have minimal and maximal sets of unchanged reachable states in the admissible models updated from the original sub model AFS1-1 as follows:
Set_{min} =updated$_1$ (AFS1-1)= $\{11(or12), 17, 18\}$;
Set_{max}=updated$_2$(AFS1-1)-{19'}
=updated$_3$(AFS1-1)$-\{20'\}$
=updated$_4$(AFS1-1)$-\{19', 20'\}$
$=\{11(or12), 17, 18, 25, 26, 15, 16\}$,
where $19'$ and $20'$ are the updated states using PU3. The admissible models with set_{max} preserve maximum unchanged reachable states in AFS1-1.

In AFS1-2, the outcome is more complex than that of AFS1-1 but has a similar principle. The 4 false states are scattered on 4 different paths. There are $(C_1^3)^4 = 81$ admissible models after update. $(C_1^3)^2 \times (2 \times (C_1^1 \times C_1^3) - (C_1^1)^2) = 45$ admissible models preserve the maximal unchanged reachable states, where $2 \times (C_1^1 \times C_1^3) - (C_1^1)^2$ is the number of combinations of different update results if PU3 is applied to at least one of states 7 or 8, then PU2, PU3 or PU5 are applied to the other state; $(C_1^3)^2$ is the combinations of different updates, PU2, PU3 and PU5, on states 23 and 24. $(C_1^3)^2 \times (C_1^2)^2 = 36$ admissible models do not preserve the maximal unchanged reachable states, which is from either PU2 or PU5 applied to states 7 and 8. The maximal unchanged reachable states for AFS1-2 are: $\{13(or14), 6, 5, 3, 4, 1, 2, 22, 21, 25, 26, 9, 10, 15, 16\}$.

For AFS1, the number of admissible models which preserve the maximal unchanged reachable states is $(2 \times (C_1^1 \times C_1^3) - (C_1^1)^2)^2 \times (C_1^3)^2 = 5^2 \times 9 = 225$.

The number of admissible models which do not preserve the maximal unchanged reachable states is $2 \times ((C_1^2)^2 \times ((C_1^3)^2)^2) - ((C_1^2)^2)^2 \times (C_1^3)^2 = 2 \times (4 \times 9^2) - 4^2 \times 9 = 504$.

Observation 1 Any admissible model obtained by applying PU2 or PU5 may not retain maximal unchanged reachable states.

IV. Minimal Change with Maximal Reachable States

Given a Kripke model $M = (S, R, L)$ and $s_0 \in S$, and let $\mathcal{M} = (M, s_0)$. We say that s' is a *reachable state* of $\mathcal{M}$, if there is a path in $M = (S, R, L)$ of the form $\pi = [s_0, s_1, \cdots]$ where $s' \neq s_0$ and $s' \in \pi$. We use $RS(\mathcal{M}) = RS(M, s_0)$ to denote the set of all reachable states of $\mathcal{M}$. Now we propose a refined CTL model update principle which can significantly narrow down the expected resulting models.

Definition 7: (Minimal change with maximal reachable states) Given a CTL Kripke model $M = (S, R, L)$, $\mathcal{M} = (M, s_0)$ where $s_0 \in S$, and a CTL formula ϕ, $Update(\mathcal{M}, \phi)$ is called *committed* if the following conditions hold: (1) $Update(\mathcal{M}, \phi) = \mathcal{M}' = (M', s_0')$ is admissible; and (2) there does not exist another resulting model $\mathcal{M}'' = (M'', s_0'')$ such that $\mathcal{M}''$ is admissible and $RS(\mathcal{M}) \cap RS(\mathcal{M}') \subset RS(\mathcal{M}) \cap RS(\mathcal{M}'')$.

The committed update preserves all unchanged reachable states in an original model after an update. The committed model results from committed update. The total set of committed models are a subset of the total set of admissible models updated from an original model. Thus, to derive committed models, we should constrain admissible update, i.e., a constraint should be added to Theorem 1 to result in Theorem 2.

A key issue of implementing the new CTL model update approach is to avoid eliminating unchanged reachable states. For this purpose, we have implemented a reachable state algorithm in code to embed the algorithm into the model updater as shown in Fig. 3.

One of the committed models of AFS1 is shown in Fig. 5, which is the result by applying PU2 to transitions before states

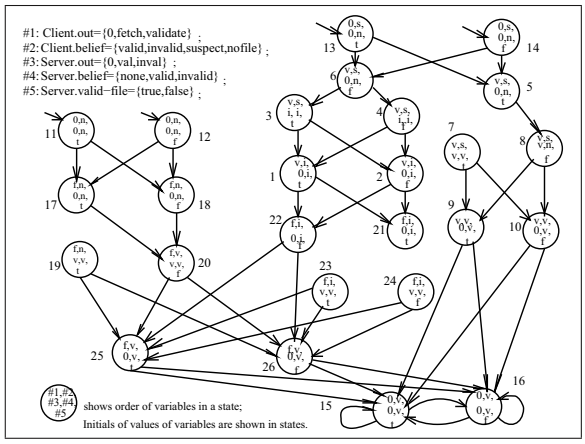


Fig. 5. One of the Committed Models of AFS1

19, 23, 24, 7 and PU3 to states 20 and 8.

V. Semantic Characterizations

According to observation 1, in order to obtain committed resulting models, any update involved in PU2 or PU5 should be carefully constrained by the reachable state principle. The characterizations for $AG\phi$, $AX\phi$, $AF\phi$ and $EG\phi$ contain primitive updates PU2 and PU5 before the constraint of Definition 7 as shown in [5]. We will further refine on some of these characterizations constrained by Definition 7 to generate more straight and simple characterizations for resulting in committed models as follows.

Theorem 2: Let $M = (S, R, L)$ be a Kripke model and $\mathcal{M} = (M, s_0)$ and $\mathcal{M} \not\models AG\phi$, where $s_0 \in S$ and ϕ is a propositional formula. Then an admissible updated model $\mathcal{M}' = Update(\mathcal{M}, AG\phi)$ can be obtained by that the following:

1) for each path starting from s_0, $\pi = [s_0, \cdots, s_i, \cdots]$: if for all $s < s_i$ in π where $s \models \phi$ but $s_i \not\models \phi$, PU2 is applied to remove relation (s_{i-1}, s_i), or PU5 is applied to remove s_i and its associated relations, iff each s_{i+1} is shared by at least another path starting from the same initial state as π, else,
2) PU3 is applied to all states s_i in π not satisfying ϕ to substitute s with $s^* \models \phi$ and $Diff(s, s^*)$ is minimal.

Theorem 3: Let $M = (S, R, L)$ be a Kripke model, $\mathcal{M} = (M, s_0)$ and $\mathcal{M} \not\models AF\phi$, where $s_0 \in S$ and ϕ is a propositional formula. $\pi = [s_0, \cdots]$ in $\mathcal{M}$ is a *valid path* of AFϕ if there exists some state $s \in \pi$ and $s > s_0$ such that $L(s) \models \phi$; otherwise, π is called a *false path* of AFϕ. A committed model $\mathcal{M}' = Update(\mathcal{M}, AF\phi)$ can be obtained by the following operations: for each false path $\pi = [s_0, s_1, \cdots]$:

1) if there is no other false path π' sharing any common state with π, then PU3 is applied to any state $s \in \pi$ ($s > s_0$) to change s's truth assignment such that $L'(s) \models \phi$ and $Diff(L(s), L'(s))$ is minimal; otherwise, this operation is only applied to a shared state s_j ($j > 0$) in maximum number of false paths;
2) PU2 is applied to remove relation element (s_0, s_1), if s_1 also occurs in another valid path π', where $\pi' = [s_0, s'_1, \cdots, s'_k, s_1, s'_{k+1}, \ldots]$ and there exists some s'_i ($1 \le i \le k$) such that $L(s'_i) \models \phi$.

Theorem 4: Let $M = (S, R, L)$ be a Kripke model, $\mathcal{M} = (M, s_0) \not\models EG\phi$, where $s_0 \in S$ and ϕ is a propositional formula. Then an admissible updated model $\mathcal{M}' = Update(\mathcal{M}, EG\phi)$ can be obtained by the following: Select a path $\pi = [s_0, s_1, \cdots, s_i, \cdots, s_j, \cdots]$ from M which contains minimal number of states not satisfying ϕ, and then

1) if for all $s' \in \pi$ such that $s' \not\models \phi$, there exist $s_i, s_j \in \pi$ satisfying $s_i < s' < s_j$ and $\forall s \le s_i$ or $\forall s \ge s_j$, $s \models \phi$, then PU1 is applied to add a relation (s_i, s_j), or PU4 is applied to add a state $s^* \models \phi$ and new relations (s_i, s^*) and (s^*, s_j);
2) $\exists s_i \in \pi$, such that $\forall s \le s_i$, $s \models \phi$; $\exists s_k \in \pi''$, where $\pi'' = [s_0, \cdots, s_k, \cdots]$, such that $\forall s \ge s_k$, $s \models \phi$, then PU1 is applied to connect s_i and s_k;
3) if $\exists s_i \in \pi$ ($i > 1$) such that for all $s' < s_i$, $s' \models \phi$, $s_i \not\models \phi$, then,
PU1 is applied to connect s_{i-1} and one of such s' to form a new transition (s_{i-1}, s');
4) if $\exists\ s' \in \pi$, such that $s' \not\models \phi$, then PU3 is applied to substitute all s' with new state $s^* \models \phi$ and $Diff(s, s^*)$ to be minimal.

Proof: In case 1, without loss of generality, we assume for the selected path π, there exist states s' that do not satisfy ϕ, and all other states in π satisfy ϕ. We also assume that such s' are in the *middle* of path π. Therefore, there are two other states s_i, s_j in π such that $s_i < s' < s_j$. That is, $\pi = [s_0, \cdots, s_{i-1}, s_i, \cdots, s', \cdots, s_j, s_{j+1}, \cdots]$. We first consider applying PU1. It is clear that by applying PU1 to add a new relation (s_i, s_j), a new path is formed: $\pi' = [s_0, \cdots, s_{i-1}, s_i, s_j, s_{j+1}, \cdots]$. Note that each state in π' is also in path π and $s' \notin \pi'$. Accordingly, we know $EG\phi$ holds in the new model $M' = (S, R \cup \{(s_i, s_j\}, L)$ at state s_0. On the other hand, we consider $Diff(\mathcal{M},\mathcal{M}')$. Clearly, $Diff(\mathcal{M},\mathcal{M}') = (\{(s_i, s_j)\}, \emptyset, \emptyset, \emptyset, \emptyset)$, which implies $\mathcal{M}'$ must be a minimally changed model with respect to $\le_M$ that satisfies EGϕ. Now we consider applying PU4. In this case, we will have a new model $\mathcal{M}' = (S \cup \{s^*\}, R \cup \{(s_i, s^*), (s^*, s_j)\}, L')$ where L' is an extension of L on new state s^* that satisfies ϕ. We can see that $\pi' = [s_0, \cdots, s_i, s^*, s_j, \cdots]$ is a path in M' which shares all states with path π except the state s^* in π' and those states between s_{i+1} and s_{j-1} including s' in π. So we also have $(M', s_0) \models$ EGϕ. On the other hand, we have $Diff(\mathcal{M},\mathcal{M}') = (\emptyset, \emptyset, \emptyset, \{s^*\}, \emptyset)$. Obviously, $\mathcal{M}'$ is a minimally changed model with respect to $\le_M$ that satisfies EGϕ.

In case 2, if all states on the first part of a path satisfy ϕ, and all states on the last part of another path satisfy ϕ, then, PU1 is applied to connect a new transition (s_i, s_k) which connects the first part of one path and the last part of the other path. Now, all states on the new path $[s_0, \cdots, s_i, s_k \cdots]$ satisfy ϕ. Thus, $\mathcal{M}' \models$ EGϕ. After PU1 is applied, $Diff(\mathcal{M},\mathcal{M}') = (\{(s_i, s_k)\}, \emptyset, \emptyset, \emptyset, \emptyset)$ is minimum and $\mathcal{M}'$ is a minimally

changed model with respect to $\leq_M$ that satisfies EGϕ.

In case 3, if PU1 is applied to form a new transition (s_{i-1}, s'), then, the new path $[s_0, \cdots, s', \cdots, s_{i-1}, s', \cdots, s_{i-1}, s', \cdots]$ contains a Strongly Connected Component (SCC) [2], [6] where all states satisfy ϕ and $Diff(\mathcal{M},\mathcal{M}') = (\{(s_{i-1}, s')\}, \emptyset, \emptyset, \emptyset, \emptyset)$ is minimum. Thus, $\mathcal{M}'$ is a minimally changed model with respect to $\leq_M$ that satisfies EGϕ.

In case 4, suppose there are n states on a path that do not atisfy ϕ, after PU3 is applied to all these states, $Diff(\mathcal{M},\mathcal{M}') = (\emptyset, \emptyset, \{s'_1, s'_2, \cdots, s'_n, s^*_1, s^*_2, \cdots, s^*_n\}, \emptyset, \emptyset)$. $Diff(\mathcal{M},\mathcal{M}')$ in this case is not compatible with case 1, case 2 or case 3. Thus, M' is a minimally changed model with respect to $\leq_M$ that satisfies EGϕ. ■

VI. Complexities

Lemma 1: Given a CTL Kripke model $M = (S, R, L)$, $\mathcal{M} = (M, s_0)$, where $s_0 \in S$, a CTL formula ϕ, and two admissible results $\mathcal{M}' = (M', s'_0)$ and $\mathcal{M}'' = (M'', s''_0)$ from the update of $\mathcal{M} = (M, s_0)$ to satisfy ϕ, checking whether $RS(\mathcal{M}) \cap RS(\mathcal{M}') \subset RS(\mathcal{M}) \cap RS(\mathcal{M}'')$ can be achieved in polynomial time.

Proof: For a given $M = (S, R, L)$, we can view M as a directed graph $G(M) = (S, R)$, where S is the set of vertices and R represents all edges in the graph. Obviously, the problem of finding all reachable states from s_0 in M is the same as that of finding all reachable vertices from vertex s_0 in graph $G(M)$, which can be obtained by computing a spanning tree with root s_0 in $G(M)$. It is well known that a spanning tree can be computed in polynomial time [8]. Therefore, all sets $RS(\mathcal{M})$, $RS(\mathcal{M}')$, and $RS(\mathcal{M}'')$ can be obtained in polynomial time. Also, $RS(\mathcal{M}) \cap RS(\mathcal{M}') \subset RS(\mathcal{M}) \cap RS(\mathcal{M}'')$ can be checked in polynomial time. ■

Theorem 5: Given two CTL Kripke models $M = (S, R, L)$ and $M' = (S', R', L')$, where $s_0 \in S$ and $s'_0 \in S'$, and a CTL formula ϕ. Deciding whether (M', s'_0) is a committed result from the update of (M, s_0) to satisfy ϕ is co-NP-complete.

Proof: Since every committed result is also an admissible one, from Theorem 5, the hardness holds. For the membership, we need to check (1) whether (M', s'_0) is admissible; and, (2) a resulting model M'' does not exist such that $(M'', s''_0) \models \phi$ and $RS(\mathcal{M}) \cap RS(\mathcal{M}') \subset RS(\mathcal{M}) \cap RS(\mathcal{M}'')$. checking whether (M', s'_0) is in co-NP [5]. For (2), we consider its complement: a resulting model (M'', s_0) exits such that $(M'', s''_0) \models \phi$ and $RS(\mathcal{M}) \cap RS(\mathcal{M}') \subset RS(\mathcal{M}) \cap RS(\mathcal{M}'')$. From Lemma 1, we can conclude that the problem is in NP. Consequently, the original problem of checking (2) is in co-NP. ■

Theorem 6: Let $M = (S, R, L)$ be a CTL Kripke model and ϕ a CTL formula. The following results hold.

1) If a committed result $Update((M, s_0), \phi)$ ($s_0 \in S$) can be obtained by only applying PU1, PU2, PU4 and PU5, then $Update((M, s_0), \phi)$ can be computed in polynomial time;
2) If ϕ is of the form $AG\psi$ and there are two states $s, s' \in S$ such that $s \not\models \psi$, $s' \models \psi$, and any path from s_0 to s' contains s, then a committed result can only be obtained by applying PU3. In this case, deciding whether an update result $Update((M, s_0), \phi)$ is committed is co-NP-complete.

VII. Conclusions and Future Work

In this paper, we demonstrated the discovery of the admissible model explosion problem for CTL model update through a running case study on the Andrew File System 1 model. We then refined our minimal change principle by retaining maximal reachable states during an update process to optimize our previous CTL model update method. We presented the implementation, semantics characterizations and computational complexity results on this improved CTL model update approach. Our work also confirmed that although in general CTL model update may generate much more resulting models than we expect, many unwanted models can actually be filtered out if we take reachable states into account.

We are considering an improvement of the reachable state algorithm as our future work. If two states are preserved in an update and there was a path between them in the original model, then there is still a path between them in the updated model. For instance, in Fig. 2, there is a path from state 21 to 26. Because these two states are preserved, there must be a path between them in all committed models. This would reduce the number of admissible models even more and would rule out the model in Fig. 5. This improved reachable state algorithm in fact provides the reachability condition from all states in a model rather than from initial states only. The reachable state algorithm could be further analyzed with graph theory.

VIII. Acknowledgements

The assistance of Senior Software Engineer Neville Coburn for the implementation of the model updater, and the support of ARC grant No. DP0664479 are acknowledged.

References

[1] Buccafurri, F., Eiter, T., Gottlob, G. and Leone, N. (1999). Enhancing model checking in verification by AI techniques. *Artificial Intelligence* 112(1999) 57-104.
[2] Clarke, E. Jr. et al. (1999). *Model Checking*, The MIT press.
[3] Cimatti, A. et al. (1999). NUSMV: a new symbolic model verifier. In *Proceedings of the 11th International Conference on Computer Aided Verification*.
[4] Ding, Y. and Zhang, Y. (2006). CTL model update: Semantics, computations and implementation. In *Proceedings of the 17th European Conference on Artificial Intelligence*(ECAI2006).
[5] Ding, Y. (2006). Model Update for System Modifications. Ph.D Thesis. School of Computing and Mathematics, University of Western Sydney.
[6] Huth, M. and Ryan, M. (2000). *Logic in Computer Science: Modelling and Reasoning about Systems*. University Press, Canbridge.
[7] McMillan,K. and Amla,N. (2002). Automatic abstraction without counterexamples. Cadence Berkeley Labs, Cadence Design Systems.
[8] Pettie, S. and Ramachandran, V. (2002). An optimal minimum spanning tree algorithm. *Journal of ACM*, 49 (2002) 16-34.
[9] Wing, J. and Vaziri-Farahani, M. (Oct.1995). A case study in model checking software. In proceedings of 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering.

Feature Modeling for Context-Aware Software Product Lines

Paula Fernandes, Cláudia Werner, Leonardo Murta
Federal University of Rio de Janeiro
COPPE - System Engineering and Computer Science
P.O. Box 68511 - Rio de Janeiro, RJ 21945-970 Brazil
{paulacibele, werner, murta}@cos.ufrj.br

Abstract

One of the first activities to develop a software product line is the feature analysis. This activity produces a feature model to represent commonalities and variabilities among products of a product line. Context-aware applications use context information to provide services and relevant information for their users. One of the challenges to build a context-aware product line is how to represent context information in a feature model. This paper proposes a modeling notation, called UbiFEX, for representing context information and defining context adaptive rules in a feature model.

1. Introduction

Context-aware systems are part of a wide range of systems within ubiquitous computing [18]. These systems use context information to provide relevant services and information to the users. In different context situations, users may access different data and exploit different aspects of an application. For instance, when a tourist arrives to a new place and accesses a mobile tourist guide application in his smartphone, he expects that recommended tours are based on his preferences and location.

Software product line paradigm [13] explores commonalities and variabilities in a set of applications for a specific domain aiming at increasing productivity and quality. It this way, it proposes a systematic software development approach based on a product family. This approach guides building new applications from reusable assets and building the assets themselves. Furthermore, this paradigm has proved itself as an efficient way to deal with varying user needs [9].

Although, most approaches have focused on the development of statically configured products using core assets with variation points [7]. All variations are instantiated before a product is delivered to customers. Hence, these approaches provide development time adaptation in which different versions of application have been generated according to customers and runtime environment specific characteristics.

For example, a mobile application such as a tourist guide is intended to execute in a variety of mobile devices and customized to different user preferences. Thus, if development time adaptation is used, the number of versions will increase exponentially. Mobile devices have limited memory, storage and processing, and it is not always possible to load at the same time all necessary components to all runtime contexts.

This static focus is not adequate to deal with the dynamism of context-aware applications because they need a runtime adaptation in which applications adapt their behavior according to context changes. Context-aware application development should benefit from the software product line concept in terms of reusability and configurability. However, it introduces new challenges to software product line engineering [16].

One of the first activities to develop a software product line is the feature analysis. This activity identifies externally visible characteristics of products in a product line and organizes them into a model called feature model [11]. A feature is a system property that is relevant to some stakeholder and is used to capture commonalities and variabilities among products in a product line.

Feature modeling allows us to model the common and variable properties of product-line members throughout the stages of product-line engineering. Feature model is used since early stages for deciding which features should be supported by a product line and which should not until product derivation. Therefore, for a context-aware product line it is important to represent context information in this model. Without this representation we can not explicitly know how this information impacts in the feature selection at runtime. We noticed that the most part of well known feature model notations [4][8] do not deal with this aspect in an effective way and they are not concerned with separating this concept for a better understanding of this kind of systems.

This paper proposes an extension to a feature notation to represent context information explicitly in a feature model. Moreover, we analyze the influence of this kind

of information in product variability and adaptive decisions. For this purpose, we define new types of features for representing context information in the model. We also define new rules to represent the relationship between features and contexts.

The remainder of this paper is divided into three sections in addition to this introduction. Section 2 reviews some basic concepts related to software product line development and context-aware systems, and summarizes some related works. Section 3 presents UbiFEX, a feature notation extension for modeling context-aware product lines. Finally, section 4 concludes this paper and discusses future directions.

2. Background

2.1. Software Product Lines

According to Software Engineering Institute (SEI), a software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy specific needs of a particular market or mission, and that are developed from a common set of core assets in a prescribed way [13]. Core assets are the essence of a product line and represent configurable elements used to build derived applications.

During the product line development process, particular aspects of products can be highlighted. The variability concept refers to points in the core assets where it is necessary to differentiate individual characteristics of products, being represented with a feature model. In order to model variability, it is necessary to represent all domain concepts and their relationships explicitly.

A feature model represents a domain and aims to make homogeneous the concepts among the participants involved in the process, such as users, domain specialists, and developers. It represents the features of a system family, their commonalities and variabilities, and the relationships among them. In addition, it has a high level of abstraction and is used as a starting point for the feature selection to new products instantiation.

An important relationship in a feature model is the dependence among features that defines when some features should be included in de product due to the presence of other features. On the other hand, it may also define when some features should be removed from the product due to the presence of other features.

This kind of model has some important concepts: (1) variation points establish the necessity of decision-making related to one feature regarding which variants will be used; (2) variants are available choices for a variation point; (3) invariants mean fixed elements that are not configurable in the domain.

For example, considering the mobile tourist guide domain, we can have a functional feature “show map”. However, maps can be represented in more than one form, such as an image or a 3-D map. In this case, we have a variation point with two variants (see Figure 1). However, we can also have a feature “list hotels” that has the same behavior for all products derived from the product line. In this case, the “list hotels” feature is considered an invariant.

2.2. Context-aware Systems

Context-aware systems are able to adapt their operations to a specific context without explicit user intervention. They use context information to provide relevant services and information.

Many definitions of context are given in the literature. Dey and Abowd [5] define context as any information that can be used to characterize the situation of an entity that is considered relevant to the interaction between a user and an application, including the user and the applications themselves.

Also, according to these authors, when dealing with context, three entities can be distinguished: places (e.g., rooms, buildings, etc.), people (e.g., individuals and groups), and things (e.g., physical objects, computer components, etc.).

In the literature [1], some attributes for describing a single context can be found: *context type* refers to the category of context; *context value* means the raw data gathered by a sensor; *time stamp* contains a date/time-value describing when the context was sensed; *source* describes how the information was gathered; and *confidence* describes the uncertainty of this context type.

2.3. Related Work

There are many approaches to support context-aware software development [12][6]. However, most of them are not concerned with a systematic software reuse. Their main focus is to solve specific problems for specific domains.

The approaches based on software product line usually propose a systematic software development based in a product family, which can beneficiate context-aware software development in terms of reusability and configurability.

Lee and Kang [10] proposed a feature-oriented approach to develop dynamically reconfigurable core assets for product lines. Features can be selected and configured at runtime. After feature analysis, feature model is refined through feature binding analysis that consists of two phases: feature binding unit identification, and feature binding time determination. Also, a dynamic binding relation is annotated with preconditions. The feature binding graph, generated after this analysis, provides an intuitive and visual description of dynamically changing product configuration.

However, mechanisms to support the definition of relevant context information used to describe preconditions are not identified and context information is not represented in the feature model.

Van der Hoek [17] presents the concept of any-time variability, which involves the ability of a software artifact to vary its behavior at any point in the life cycle. According to the author, a solution to support this kind of approach has to provide four main functionalities: a representation to capture system variabilities; a tool to specify these variabilities; a tool to resolve variabilities; and tools to apply the result of this resolution, in different life cycle points. This work does not exploit the way how the context information is identified. Furthermore, on the variabilities graphic representation, there are not elements which identify how this information influences the system dynamic configuration.

The approach proposed in this paper aims to provide solutions to the weak points found in these works, allowing an explicit representation of the relevant context information to the domain in the feature model and identifying how this information influences on the system dynamic configuration.

3. UbiFEX

UbiFEX is proposed to be a feature notation that provides context information representation and context rules specification. UbiFEX extends Odyssey environment [15] feature notation, called Odyssey-FEX.

This section is divided in four subsections. Section 3.1 presents the main characteristics of Odyssey-FEX notation, used as base for the proposed extension. Section 3.2 describes the features categories defined to allow context representation. Section 3.3 and 3.4, respectively, present expressions and rules used to represent the context influence in the dynamic product configuration. Finally, Section 3.5 introduces the initial ideas to build a context simulation infrastructure.

3.1. Odyssey-FEX

The Odyssey environment provides support to software reuse through domain engineering, product lines and component based development techniques.

Odyssey-FEX [14] was developed to fill some gaps in variability representation detected in other feature notations that could lead to an incorrect modeling of a system family. Some of these gaps are the lack of an explicit representation of variation points and an insufficient representation of dependency and mutual exclusiveness relationship among features.

In this notation, features must be represented according to three dimensions: category, variability, and optionality. There are five categories representing feature types: domain (functional and conceptual), entity, operational environment, domain technology, and implementation techniques.

Domain features are related to the core domain functionalities and concepts. Entity features are the model actors. Operational environment features represent attributes of an environment that a domain application can use and operate. Domain technology features represent technologies used to model or implement a specific domain requirement. Finally, implementation techniques features represent technologies used to implement other features.

According to variability classification, features can be variation points, variants or invariants. These concepts were previously presented in Section 2.1.

In respect to optionality, a feature can be mandatory or optional. This classification indicates whether a feature should be present in all products or not. The optionality refers to the whole domain. In Odyssey-FEX notation, optional features are represented in the model with a dashed shape. It is important to notice that an optional feature may become mandatory when other features are selected. This situation may occur due to the existence of composition rules among those features.

Composition rules define restrictions between features. Odyssey-FEX defines two types of composition rules: inclusive and exclusive. Inclusive rules represent feature dependency. For example, when the antecedent is selected the consequent has also to be selected. Exclusive rules represent mutually exclusive feature relationship. In this case, when the antecedent is selected the consequent must not be selected for the same product. These rules can be combined with boolean expressions. These expressions can be composed by more than two features, forming an expression combination both for a rule antecedent and consequent.

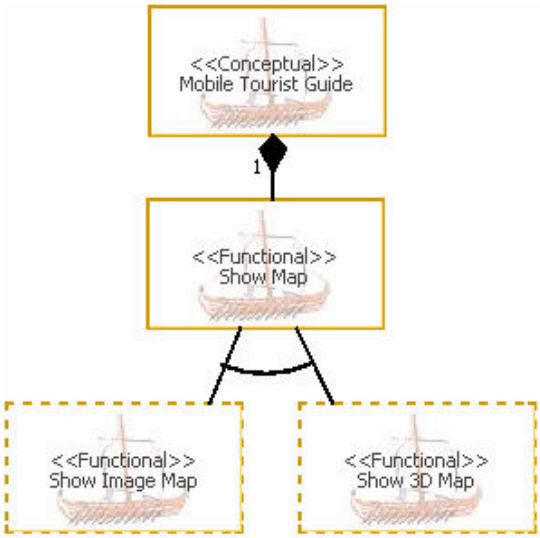


Figure 1. An example of Odyssey-FEX notation.

Odyssey-FEX applies relationship semantic in a feature model, offering a stronger capacity of representation and expression. Features are related to each other using UML relationships, such as association, generalization, and composition. In addition, the notation links a variation point to their variants through alternative relationships. This relationship expresses variability in the feature model.

Figure 1 shows a part of a feature model to a mobile tourist guide domain [3] built using Odyssey-FEX.

Odyssey environment also supports other feature model notations, including the ones proposed by Gomaa [8] and Czarnecki [4]. We decide to extend Odyssey-FEX due to its high level of expressiveness in respect to the other notations.

3.2. New Feature Categories

UbiFEX is proposed to represent context information in an explicit form. For that purpose, we defined two feature categories in addition of those described in the previous section: context entity and context information.

Context entity feature was created to represent relevant context entities for the domain. This relevance is based on the influence of the entity on the system behavior. For example, the mobile tourist guide system has entities such as user, mobile device, or a specific environment. The properties defined to a context entity feature are: name and description.

Context entities can be characterized by context information. Context information feature represents the data that should be collected to describe a context entity, which are relevant to domain applications adaptation.

Based on the attributes described in Section 2.2, we defined a set of properties to this type of feature: name, description, type (i.e., static or dynamic), base type (e.g., string, integer, etc.), initial value (when applicable), and source. Some of these properties are application specific, thus, they can be filled in during the domain analysis or after the initial feature selection to derive an application, according to the modeler decision.

If context features are modeled together with other feature types, the final variability model may be polluted with different concerns, compromising understandability. For this reason, we recommend the use of a separated model for context feature modeling. In this case, relationships between the models are represented by context rules. Odyssey environment allows filtering features in different types of diagrams according to their categories. In this way, we may have different views of a single feature model.

Figure 2 shows a simple context model to mobile tourist guide domain, representing the entities and context information.

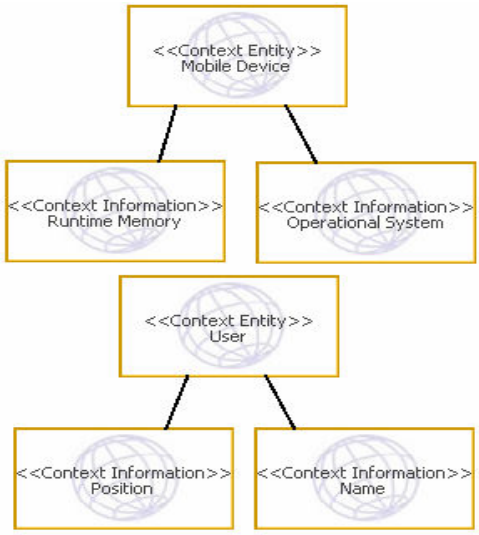


Figure 2. A context feature model.

3.3. Context Definition Expressions

After modeling context entities and information, the next step is the definition of the contexts that are necessary to create the context rules. Context definition is composed by a name and an expression.

```
<context-definition> ::= <expression>

<expression> ::= <CIF><relational-operator> <value>
        | <expression> <logic-operator> <expression>
        | NOT <expression>

<relational-operator> ::=  > | < | >= | <= | = | <>

<logic-operator> ::=  AND | OR | XOR

<value> ::= <string type> | <int type> | <float type>
        | <boolean type>
```

Figure 3. BNF for context definition.

A context can be defined according to the BNF notation (see Figure 3). An expression can be formed by a context information feature (<CIF>), previously described in the feature model, a relational operator, and a value. Also, an expression can be a composition of expressions using logic operators.

Figure 4 illustrates an example of a context definition expression for the mobile tourist guide domain.

(Runtime Memory > 64) **AND**
(Operational System = “Symbian”)

Figure 4. Example of context definition expression.

In this way, we can define a context named Enough Memory associated with the first expression. A context is active when the evaluation of its expression is true.

3.4. Context Rules

Context rules specify how a specific context affects an application configuration in the domain, determining, for example, the decision about variant selection in a variation point.

The construction of a context rule is similar to a composition rule. The rule is formed by an antecedent, the operator *implies*, and a consequent. The antecedent is an expression that can contain contexts, features, and logic operators. The operator *implies* means that if the antecedent is true, then the consequent is selected. The consequent is an expression that can contain features and logic operators. For the domain used in this paper, Figure 5 shows an example of a context rule.

This rule can be used to optimize functionalities. For example, when the tourist wants to visualize a city map, the application analyses the active contexts and rules and choose the better variant to the runtime context.

Enough Memory **implies** Show 3D Map

Figure 5. Example of a context rule.

The context feature model with context definitions and context rules can be exported to an XML file. This file can be used as input to generate, for example, a middleware configuration file. These rules are also represented in the feature model. Features that are part of the consequent in some rule are marked with the rule identifier. In this way, it is easier to identify which features have been influenced by context.

3.5. Context Simulation

Starting from these proposed extensions, we are working on a tool to simulate variations in the defined contexts to analyze the behavior of the product line architecture. The Odyssey environment domain engineering process works with component-based architectures and it is possible to map features to application components.

In this way, we can identify inconsistencies between composition rules and context rules. The advantage is that it can be done at development time, reducing the cases where there is no possible product configuration and the application will probably fail.

For example, consider the previous mentioned mobile tourist guide domain. The simulation follows the following steps. First, we have to choose an initial configuration for the product, selecting the initial features. After that, we can simulate variations in the context information features values, such as Runtime Memory, and determine which contexts are active based on context definition expressions. The next step is the context rules analysis, to check if there are modifications on the product configuration. If modifications are found, a new set of features is selected and the consistency of composition rules is checked. For a complete simulation, these steps have to be repeated for each new generated configuration.

This simulation can run on-line, immediately notifying inconsistencies, or in batch mode, enacting distinct scenarios and generating reports with all existing inconsistencies.

4. Conclusions and Future Work

This paper presented a feature model notation to context-aware product lines called UbiFEX. Some extensions are proposed to an existing feature modeling notation, named Odyssey-FEX, to support representation of context information and dynamic product derivation. In this way, UbiFEX proposes a solution for the weak points found in the related work regarding context representation, providing a better understanding and representation of the relevant context entities and information in a context-aware domain, using the same type of model to represents other feature types.

That explicit representation of context is essential to modeling context-aware applications. Contexts influence directly the behavior of these applications. Therefore, it is important to identify them since the first product line development activities.

UbiFEX also promotes a first step to a dynamic configuration of products based on context rules. Moreover, it allows an early verification of the product execution through context simulation.

Context information is a key element to produce self-adaptive applications in ubiquitous computing [2]. For this reason, our first concern was the way of dealing with this type of information in the software product line development. However, there are still many challenges to build context-aware product lines, mainly due to the dynamic adaptation aspect of this class of applications.

As future work, we are planning to evaluate the proposed feature notation in a real scenario, modeling a family of existing context-aware applications for mobile devices. Then, we can analyze if relevant contexts and adaptive rules can be represented in an effective way. Since Odyssey environment supports other feature model notations, we intend to estimate the effort to extend the proposed approach to these notations.

Moreover, we also intend to use feature models to dynamic product derivation, developing an automated product variant selection based on features and rules proposed in this paper.

5. Acknowledgments

The authors would like to thank CAPES and CNPq for the financial support.

6. References

[1] Baldauf, M. and Dudstar, S. 2004. A survey on context-aware systems. Tech. Report TUV-1841-2004-24, Tech. Univ. of Vienna.

[2] Bardram, J. E. 2005. The Java Context Awareness Framework – A Service Infrastructure and Programming Framework for Context-Aware Applications, Third International Conference, Pervasive2005, Munich, Germany, 98-115.

[3] Baus, J., Cheverst, K., and Kray, C. 2005. A survey of map-based mobile guides. Map-based mobile services - Theories, Methods, and Implementations, 197-216.

[4] Czarnecki, K., Helsen, S., and Eisenecker, U. 2004. Staged Configuration Using Feature Models. In Proc. of Software Product Line Conference (SPLC 2004), Lecture Notes in Computer Science, Springer-Verlag, 266-283.

[5] Dey, A. 2001. Understanding and Using Context. *Personal Ubiquitous Comput.* 5, 1 (Jan. 2001), 4-7.

[6] Garlan, D., Siewiorek, D., Smailagic A., and Steenkiste, P. 2002. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive Computing* 1, 2 (April 2002), 22-31.

[7] Gomaa, H. and Hussein, M. 2003. Dynamic Software Reconfiguration in Software Product Families. In Proc. of the 5th Int. Workshop on Product Family Engineering (PFE), Lecture Notes in Computer Science, Springer-Verlag, 435-444.

[8] Gomaa, H. 2004. Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures, Addison-Wesley Professional.

[9] Hallsteinsen, S., Stav, E., Solberg, A., and Floch, J. 2006. Using Product Line Techniques to Build Adaptive Systems. In *Proceedings of the 10th international on Software Product Line Conference* (August 21 - 24, 2006). International Conference on Software Product Line. IEEE Computer Society, Washington, DC, 141-150.

[10] Lee, J. and Kang, K. C. 2006. A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering. In *Proceedings of the 10th international on Software Product Line Conference* (August 21 - 24, 2006). International Conference on Software Product Line. IEEE Computer Society, Washington, DC, 131-140.

[11] Lee, J. and Muthig, D. 2006. Feature-oriented variability management in product line engineering. *Commun. ACM* 49, 12 (Dec. 2006), 55-59.

[12] McKinley, P. K., Sadjadi, S. M., Kasten, E. P., and Cheng, B. H. 2004. Composing Adaptive Software. *Computer* 37, 7 (Jul. 2004), 56-64.

[13] Northrop, L. M. 2002. SEI's Software Product Line Tenets. *IEEE Softw.* 19, 4 (Jul. 2002), 32-40.

[14] Oliveira, R. 2006. Formalization and Consistency Checking in Variabilities Modeling. Master Thesis. Federal University of Rio de Janeiro, Rio de Janeiro, Brazil (in Portuguese).

[15] Software Reuse Team. 2008. Odyssey Project. http://reuse.cos.ufrj.br/odyssey.

[16] Sugumaran, V., Park, S., and Kang, K. C. 2006. Introduction – Software product line engineering. *Commun. ACM* 49, 12 (Dec. 2006), 28-32.

[17] van der Hoek, A. 2004. Design-time product line architectures for any-time variability. *Sci. Comput. Program.* 53, 3 (Dec. 2004), 285-304.

[18] Weiser, M. 1999. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.* 3, 3 (Jul. 1999), 3-11.

MEtaGile: A Pragmatic Domain-Specific Modeling Environment

Olivier Buchwalder, Claude Petitpierre

Networking Laboratory
Swiss Federal Institute of Technology in Lausanne
CH-1015 Lausanne EPFL, Switzerland
E-mail: {olivier.buchwalder, claude.petitpierre}@epfl.ch

Abstract

Domain-specific modeling (DSM) is a software development methodology that promises greater gains in productivity by systematizing the use of domain-specific languages (DSL). This paper first addresses the notions of abstraction and specificity by comparing some existing languages, and proposes an original representation that highlights the global advantages of using DSLs. This document presents then MEtaGile, our DSM environment that provides facilities for creating and supporting evolved DSLs. This environment is mainly designed for supporting pragmatic modeling concepts, and implements practical features for supporting the code-generation phase. The development of DSLs is facilitated by the use of a simple but efficient meta-language that allows the domain-specific developers to focus on the final model-to-text transformation; they are neither expected to be expert in modeling nor to master complex transformation languages.

1. Introduction

The software industry is under high pressure to reduce the cost and the development time of the applications, but the global complexity of modern applications increases. In comparison with other engineering branches, such as the automotive industry, the software industry lacks automation, which requires precise models that abstract the system structure and behavior by hiding non-relevant details. The development methods that integrate the models are qualified as Model Driven Development (MDD [1]). The domain field of a general modeling language such as UML is too large for defining precisely the domain-specific concepts needed for the generation, and is mainly used in practice for documentation or discussion [3, 15, 17]. The extension facilities that have appeared with UML 2.0, for supporting MDA [9], make it possible to define specific concepts [5], but UML is already a complex language and many developers have difficulty to use it efficiently [18]. The domain-specific modeling (DSM [16]) approach, which follows the MDD principles, attempts to reduce the gap between the model and the concrete system by using domain-specific languages (DSL [12]). A DSL provides a specialized semantic, which increases the model precision. In order to be used in practice, a DSL needs a compiler and an adapted CASE tool for helping and guiding the developers during the designing process of the system instances. A DSM environment (Meta-CASE tool or DSL tool) is also needed to simplify the creation of the DSL compiler and of the associated tool. This paper presents MEtaGile, a DSM environment integrated in Eclipse, which provides the required support for defining a DSL and an adapted tool, such as facilities for editing, visualizing and validating the domain-specific models and for automating the generation of end-user systems.

This paper is organized as follows: Section 2 addresses the DSM approach and proposes an original representation of the language properties. Section 3 presents the main features of the MEtaGile environment, and Section 4 addresses the DSL development and presents a simple example.

2. Domain-Specific Modeling

DSM is a MDD approach that represents a viable solution for increasing the productivity of application development [4, 6, 16]. This approach proposes the use of DSLs for modeling systems in a specific domain instead of using general-purpose language (GPL); the DSL model represents at the same time the design, the implementation and the documentation of the system.

An important benefit of DSM results from the splitting of the development process in two successive phases, which can be addressed by different groups of developers. The first phase, the DSL definition must be realized by experts of the domain, and the second phase, the system design can be handled by most developers; the tool is intended to support

the human designing work, and to automate the transformation to concrete code. This approach avoids the stiffness of the traditional development environments; a company can manage its own DSL adapted to its specific needs, and be free to rapidly evolve its definition without external constraints.

2.1. Analysis of the Domain-Specificity

The current section presents an original 2D graph that highlights the benefits of DSLs, and shows how the productivity of a given development is relative to the nature of the language (meta-model) used to develop the system. The global development productivity is positively influenced by the abstraction level (axis x) and by the specificity (axis $-y$) of the language (Figure 1). The abstraction level represents the capacity of a language to hide the concrete details of a system in providing high-level concepts. The specificity level is directly linked to the domain range that a language is able to address. Both notions are not precisely quantifiable, because a language can include different aspects more or less abstract, or specific. However, the global properties of languages can be approached by using 2D areas.

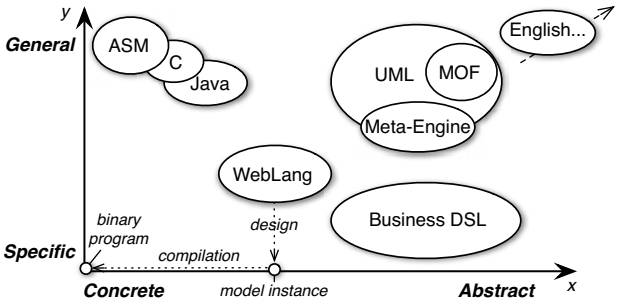


Figure 1. Classification of languages

The presented graph highlights the needed phases for developing an executable system by using different languages. The design phase, represented by the vertical transformation to the bottom axis, is mainly dedicated to the developers, which use the language to create a specific model that implements the system specific requirements. This phase is not automatable along the development of different systems in the same domain. Figure 1 shows that the use of a DSL, close to the business domain of the system, highly reduces the developers intellectual work for creating a specific model instance.

The compilation phase is represented by the horizontal transformation from the specific model instance to the concrete executable code; this process is usually automated by a compiler tool. When a language is highly general and abstract, as is the case of UML, both transformation phases are difficult, and the development of a system is not efficient enough [6, 12].

3. The MEtaGile Environment

The environment we have developed to support the DSM approach provides assistance for creating a new textual DSL or meta-model, adapted to a specific domain (Section 4), and also supports the developers that use the defined DSL to instantiate systems (DSL instance or model).

In comparison with others solutions, such as OAW [14], Microsoft DSL tools [6], GME [11], or MetaEdit+ [19], MEtaGile handles pragmatic development aspects, such as the unique textual model for input, the hierarchical and graphical views for documentation and navigation, the redefinition in real-time of the DSLs and of the templates, or the efficient management of the generated files.

3.1. Extended Textual Modeling

MEtaGile is exclusively based on centralized textual models for editing concerns, which are heavily supported by an evolved generic editor. This editor provides for any DSL models the syntactic highlighting, an editing assist feature, the displaying of the validation messages. It also integrates read-only views, which allow the developers to visualize abstractions of the system, and to navigate efficiently in the textual model. An outline view presents the hierarchical tree of a given model, and diagram views provides graphical representations of the model. MEtaGile enables the presentation of several diagrams that are related with one source model, and thus supports the separation of concerns for documentation and navigation. However, we are convinced that the editing of a complex model is more efficient using a centralized and textual model that fully defines the target system; a precise diagram usually requires the insertion of hidden textual constraints or action language, which are not easily representable using a graphical notation. This centralization enables the simplification of the model-to-text operation, because only one model source is used as input. The use of simple text files also allows the developers to work with the models, and to use efficient and largely adopted textual functionalities, such as the universal $copy/paste$ or the CVS sharing; a textual model is also naturally and efficiently editable with the keyboard.

Only the Xtext OAW component also enables the definition of text-to-model transformations, but using MEtaGile the definition of the parser and of the structural concerns of the meta-model are centralized in a unique language. Moreover, our extended BNF syntax is based on JavaCC [7], and allows the handling of more complex expressions, which can also be defined with the Java language.

3.2. Redefinition Capabilities

MEtaGile integrates some features that address specifically the code-generation phase (model-to-text), and of-

fers to the developers a way to support efficiently the successive generations of the target system files, and certain de-synchronization between the model and the produced files. Our code-generation feature uses the template technology JET [8], which is close to the JSP implementation of Apache Tomcat.

3.2.1. Engine Pluggability. MEtaGile is composed of Eclipse plugins that handle the generic features, such as the editor and the views, and of DSL engines, which contain all domain-specific properties. Contrary to the plugins, the modifications applied to the engine are effective in real-time without having to restart the Eclipse platform. This loosely coupled architecture enables the developers to switch easily between different versions of engines, to develop and test in parallel DSL engines and instantiated models, and therefore to increase the development productivity. The other DSM tools are heavier to deploy, and usually require restarting the entire system after a modification. This feature is implemented using a redefined Classloader that addresses the original static classes of the Eclipse plugins and the dynamic classes of the active engine; this dynamic Classloader is used to execute the parsing operation defined in the DSL engine. The generic environment accesses domain-specific information of a model, using the Java reflection and the meta-model defined in the relative DSL engine.

3.2.2. De-Synchronization. When the development addresses the target system details, many manual modifications of the generated system files are often required, and must be preserved. For addressing this purpose MEtaGile enables the developer to visualize and select the replacement mode of the produced files; a report window displays the responsible source model element, the local output path, and the replacement mode. The available modes are *createonly, overwrite, deactivated* and *merge*; the latter is currently available for the Java files using the JMerge JET technology, and a 3-way merging method is already included for the XML files. This merging technique is simple and flexible enough to support a limited de-synchronization between the model and the output files. The generation mode is stored permanently for each file and can be shared using a CVS server; this feature enables the developers that are not involved in some specific modifications to regenerate the whole system without overwriting some important files or file parts.

3.2.3. Specific Template Redefinition. For addressing the specificities of an application instance, our environment is capable of handling local redefinitions of the templates. This feature offers a flexible way to include application specific properties rapidly and efficiently, in keeping the model and the generated application synchronized. This redefinition is activated by using a specific annotation in front of a model element or globally at the beginning of a model file. The annotation statement is $@templatedir = package$, where package represents the path where the redefined template class will be emitted. A template file is always relative to a specific node type, and by using similar package identifiers, two instances of the same type can share the same redefined template. MEtaGile provides an operation that loads and prepares the redefined templates into the user project. The application developer can freely modify the content of the templates; the whole model data is accessible using getter functions.

Only OAW currently provides a similar feature, but the redefinition of templates using MEtaGile is in our opinion easier and more flexible. Our templates are modifiable without having an important knowledge about the environment as expected with the OAW approach, and the latter doesn't allow different model elements to use different redefined templates.

4. DSL Engine Modeling

Our solution supports the DSM approach, including the development of the DSL engines using a meta-DSL engine that includes a meta-meta-model. This meta engine allows the domain experts to define and produce a valid DSL engine, which can be used by other developers to define systems in a specific domain. In comparison with other environments, our approach attempts to propose a minimal but simple and centralized way to define and generate the meta-model, the main transformation from text-to-model, and the foundation for the model-to-text transformation, as expected in most cases.

Other approaches usually require defining separately the structural model, the transformations and validation rules, and how these processes are linked together. The definition of a similar DSL tool will require much more effort and knowledge using OAW than using MEtaGile for most cases. Indeed, OAW or other approaches use advanced but heavy languages for specifying transformation and the validation rules, such as QVT, ATL and OCL [13, 10]. Mastering these languages and defining evolved model-to-model transformation and validation rules can improve the quality and the abstraction of the process, but it requires an important investment, and is not adapted with all transformation forms. For instance, when a transformation intends to create output model elements that are not directly related with easily identifiable input model elements, or depends on many different elements, a transformation specified with a Java-like language can be more suitable.

Our meta-DSL, which represents the meta-meta-language of a concrete system, supports bootstrapping [1]. This language is also able to define a full engine component that contains the structure of the meta-model, the basic validation and transformation rules including the parser proper-

ties, which represents the text-to-model process. The structural entities of an engine are specified using a meta-model that includes the principal object-oriented aspects, such as encapsulation, modularity, polymorphism, and inheritance, but it provides specific terms and concept relative to the engine specification. The defined structure is designed for supporting a high modularity; the definition of the checking, processing and producing functions are located in the relative node elements; these entities are qualified as module for main entities and submodule for children elements. Modules and submodules include fields that are intended to specify referenced entities or values. Each node component is susceptible to produce output files, dynamically from a template or by copying a resource file.

MEtaGile also supports the definition of extended graphical views of the model for documentation and navigation concerns; the meta-language includes concepts for creating a view with a selection of modules and sub-fields, and also allows the creation of new elements or sub-element that are not directly related to source model elements (model-to-model transformation). Other DSM environments that support model transformations could also define equivalent views, but the advantage of our approach is to integrate and synchronize the extended views naturally and efficiently in the DSM tool.

4.1. Example of Engine Modeling

This Section presents a realistic example of the definition of a DSL-engine that describes a simple DSL. The target domain of the DSL, presented in the following model, represents a hierarchical web site composed of pages that contain articles and links to other pages. This model defines the various DSL engine properties: the structural elements, the parser syntax and the producing templates. The $page$ entity is defined as a main module that includes fields for hosting the relative properties, as the name, the description and the header. The $articles$ field list references the articles that are defined locally; an article is defined as a local submodule of the page, and includes simple typed fields, filled by the local parser. The page element includes the list $links$ that contains the referenced page names, and the list $pageLinks$ that contains the referenced page modules. This last list enables the navigation in the page hierarchy more easily, but it is not automatically filled by the parser and must be populated by the developer in the processing method of the page; this population logic is quite trivial: an iteration of all page links is included in another iteration of all defined page instances, the page is added to $pageLinks$ when its name equals the link identifier. Two templates are defined in the page element declaration; the first one is responsible to create an output html file for each page instance, and the second static template attempts to create a unique read-me file that includes a reference on each available page.

```
module Page {
  mainkeyword = "page";
  template = ( page.html, name + ".html");
  templatestatic=(readme.htm,"doc.html","doc");
  String name, head, descr;
  list<Article> articles;
  list<String> links;
  list<Page> pageLinks;
  parser {
    name "{"
      "heading" head = STRING ";"
      ["description" descr = STRING ";"]
      ["links" "=" links ("," links)* ";"]
      ( articles )*
    "}"
  }
  submodule Article {
    String title, content, authorId;
    boolean isFinished = false;
    int nbWord;
    parser {
      "article" ["finished" isFinished:=true] "{"
        "title" "=" title = STRING ";"
        "content" "=" content = STRING ";"
        ["words" "=" nbWord ";"]
      "}"
} } }
```

After having generated this specific DSL engine, using the meta-DSL engine, the developer must edit the prepared JET templates and introduce the domain implementation details, here the html code with the dynamic accesses to the page and article properties. Then, system instances can be defined and generated in using the newly generated DSL engine, such as the simple Web-site example specified by the following model.

```
page Index {
  heading "My Watch Company";
  description "Since 1872";
  links = Collections, Sponsoring;
  article finished {
    title = "Happy New Year 2008";
    content = "Our company is happy to...";
    words = 200;
} }
page Collections {
  heading "Collections 2008";
  description "The new Collection is...";
  links = Sport, Classic;
  article {
    title = "A specific Watch for...";
    content = "...";
} }
page Sport {...}  page Classic {...}
```

Figure 2 presents the graphical view output of the given model; each element is displayed as a box, the inner submodules are by default folded and displayed in a list, but the user can change these view properties, as the presented $Classic$ page, where the articles are presented in a 2D layout. Each element or sub-element that includes a reference on another element is displayed by default in a 2D layout,

and the reference as a link; a link between folded elements is reported to the respective parent and visible element.

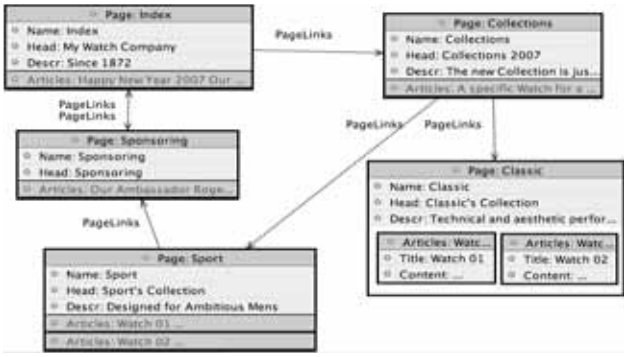


Figure 2. Generic Graphical View

4.1.1. Practical Tests. The previous example is deliberately simple, but the presented meta-DSL is able to specify evolved DSLs as the meta-DSL itself or WebLang [2], which attempts to design the architecture of J2EE Web-applications in a concise and efficient way. It enables the definition of elaborated application models by assembling different component instances, and then producing executable applications, in abstracting the implementation details (see Figure 1). It has been successfully used for three years by hundreds of students of the EPFL for supporting the software engineering course.

Others DSLs have been developed: a Java3D DSL that is able to define evolved 3D scenes in abstracting the framework details, and a PHP DSL able to create evolved sites composed of dynamic pages connected to a database. A DSL, able to define and generate .Net Web sites that includes structural and behavioral concerns, has been developed using the Microsoft DSL tools (2005), and using MEtaGile. This exercise has shown that the Microsoft DSL tools are currently not well adapted to the generation of final applications; the use of MEtaGile and of our textual modeling approach allows the developers to save time in the development of the DSL. The definition of graphical elements requires developing advanced wizards for conducing the developers to set the mandatory properties correctly, but the DSL tools don't offer an efficient support for this task.

5. CONCLUSION

This paper has first discussed abstraction and specificity concerns of some existing languages, and has shown how the combination of these notions influence positively the development productivity. The use of a DSL as a modeling language allows the designers to efficiently manipulate domain-specific concepts; the automation process is also optimized by a precise model and by the specialization of the target domain. The current paper has also presented MEtaGile, a DSM environment, that provides facilities for creating and supporting evolved textual DSLs. This environment is mainly designed for supporting pragmatic programming, and implements practical features for supporting the code-generation phase; it integrates a loosely coupled architecture that supports rapid DSL evolutions, and a template redefinition functionality, which enables the DSL users to easily adapt some templates for a specific use. The use of a simple but efficient meta-language allows the domain-specific developers to efficiently define textual DSLs; they are not expected to be expert in modeling, and to master transformation and validation languages. Further development will address the definition of DSLs that address more specific business, such as the management of projects, stock, or customers.

References

[1] C. Atkinson and T. Kuehne. Model-driven development: a metamodeling foundation. *IEEE Software, Volume 20, Issue 5, Sept.-Oct. 2003 Page(s):36 - 41*, 2003.

[2] O. Buchwalder and C. Petitpierre. WebLang: A Language for Modeling and Implementing Web Applications. In *SEKE06*, 2006.

[3] M. Fowler. *UML distilled: A brief Guide to the Standard Object Modelling Language*. Object Technology series. Addison Wesley, 3rd edition, 2004.

[4] M. Fowler. Language workbenches: The killer-app for domain specific languages? www.martinfowler.com, 2005.

[5] E. Gorshkova and B. Novikov. Exploiting uml extensibility in the design of web applications. http://citeseer.ist.psu.edu/530237.html, 2003.

[6] J. Greenfield and K. Short. Software factories: assembling applications with patterns, models, frameworks and tools. In *OOPSLA '03*, pages 16–27. ACM Press, 2003.

[7] JavaCC. https://javacc.dev.java.net/.

[8] JET. http://www.eclipse.org/modeling/m2t/?project=jet.

[9] A. Kleppe, J. Warmer, and W. Bast. *The Model Driven Architecture-Practice and Promise*. Addison-Wesley, 2003.

[10] I. Kurtev, K. van den Berg, and F. Jouault. Evaluation of rule-based modularization in model transformation languages illustrated with atl. In *SAC '06*. ACM, 2006.

[11] A. Ledeczi, M. Maroti, A. Bakay, and G. Karsai. The Generic Modeling Environment. In *Workshop on Intelligent Signal Processing, Budapest, Hungary*, May 2001.

[12] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM*, 2005.

[13] OMG. *MOF QVT Final Adopted Specification*. Object Modeling Group, June 2005.

[14] openArchitectureWare. http://www.eclipse.org/gmt/oaw.

[15] B. Rumpe. Executable modeling with uml. a vision or a nightmare? *Issues and Trends of IT Management*, 2002.

[16] J.-P. T. Steven Kelly. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE, 2008.

[17] D. Thomas. Uml - unified or universal modeling language? *Object Technology, vol. 2, no. 1, January-February*, 2003.

[18] D. Thomas. Mda: Revenge of the modelers or uml utopia? *IEEE Software, vol. 21, no. 3, pp. 15-17*, 2004.

[19] J.-P. Tolvanen and M. Rossi. Metaedit+: defining and using domain-specific modeling languages and code generators. In *OOPSLA '03*. ACM, 2003.

Obtaining Well-Founded Practices about Elicitation Techniques by Means of an Update of a Previous Systematic Review

Oscar Dieste
Facultad de Informática
Universidad Politécnica de Madrid
28660 Boadilla del Monte, Spain

Fraunhofer IESE
Fraunhofer Platz 1
67663, Kaiserslautern, Germany

odieste@fi.upm.es
oscar.dieste@iese.fraunhofer.de

Marta López
Facultad de Informática
Universidad Complutense de Madrid
C/Jose García Santesmases S/N
28040 Madrid, Spain

mlf@fdi.ucm.es

Felicidad Ramos
INDRA Systems
Carretera Loeches 9
28850. Torrejón de Ardoz, Spain

framos@indra.es

Abstract

Several studies point out that elicitation techniques achieve different results when applied in different contexts. This paper presents some recommendations about the situations in which elicitation techniques are useful. Recommendations are based on a previous systematic review, which was updated and expanded with 13 new empirical studies and more than 60 new empirical results. The aggregation process generated 5 new evidences and modified 4 existing ones. In the previous review, it was found that interviews were one of the most adequate techniques in most situations. The new evidence supports the same conclusion.

1. Introduction

Nowadays it is widely acknowledged within the software engineering community that requirements definition has a big impact on final product quality[10-12]. Requirements Engineering (RE) is concerned with the elicitation, analysis, specification, validation and management of software requirements [13]. This paper focuses on the elicitation task and, more concretely, on the techniques applied to extract knowledge from the requirements stakeholders.

Although requirements elicitation appears to be a simple process in fact it is a really difficult one. Quite often, users do not know how to describe their tasks, may leave important information unstated, or may be unwilling or unable to cooperate [13]. Elicitation techniques aim to improve this communication process.

Despite the critical need for eliciting the right requirements, little research had been focused on identifying the most adequate elicitation techniques. Only ACRE[14] and recently the Unified Model of Requirements Elicitation[15,16] provide general frameworks. However, these works are by and large rooted on quite general theoretical foundations or expert opinion, leaving aside an increasingly large body of empirically-based knowledge.

Systematic Review (SR) is a technique employed in Evidence-Based Software Engineering (EBSE)[17], whose aim is to pool together the results obtained in different empirical studies and propose recommendations based on the best available evidence. In a previous work[18-20], the authors shown that SRs are a useful way to identify good practices regarding requirements elicitation. 30 different empirical studies were identified, reviewed and aggregated, generating 18 evidences about interviews, protocol analysis, sorting and laddering techniques.

A critical fact in any SR is the amount of evidence available. SR's conclusions are always based on the existing evidence when the SR is done, but as new empirical studies are discovered (because they were not identified before) or carried out, the conclusions of earlier SRs should be updated, either confirming or refuting the previous findings. This present paper updates the previous SR adding 14 new empirical studies and more than 60 empirical results. The subsequent aggregation process generated 5 new evidences and modified 4 existing ones. The new and modified evidences are in line with those in [20]. Generally speaking, they point out that interviews are

the most effective elicitation technique in most situations, although its efficiency may be lower than some specialized techniques like laddering or card sorting in some cases.

This paper reports how the update of the previous SR was carried out and which evidences were obtained. It is structured as follows: Section 2 briefly describes the research methodology. The main findings are shown in section 3. Finally, in section 4 we discuss the findings and enumerate the main conclusions.

2. Methodology

This work is an update of the SR described in [20]. That SR was carried out following the recommendations proposed in [21]. In this first SR, 53 publications containing potential empirical studies were identified. An initial search in online repositories and local library resources made possible to obtain 26 of those publications. These 26 publications contained 30 empirical studies which were reviewed and aggregated as already mentioned.

However, 27 potentially interesting publications were disregarded. As the conclusions of the SRs are contingent upon the available evidence, it was clear that a more thorough search (e.g. in international library services) was desirable. It made possible to obtain 13 out of those 27 publications. The other 14 publications (e.g. [22,23]) were considered impossible to locate, as they are quite old, grey literature. The overall literature flow is shown in Figure 1.

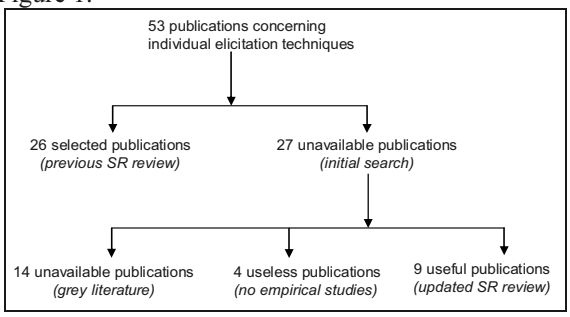


Figure 1. SR literature flowchart

Not all those 13 publications were useful. Four of them did not contain empirical studies at all or were papers published twice, so that they were discarded. The other nine[1-9] were useful and gave 13 empirical studies ([4] contained 2 different studies while [6] contained 3). The current updating work focused on those 13 empirical studies.

The tasks performed so far correspond to the initial stages of Kitchenham's procedure[24]. In a typical SR, both the review objective and the identification of studies would have been part of the SR itself. However, in this concrete case, we are making an update of a existing SR (that is, the reference [20]) and therefore these tasks are obviously skipped.

The subsequent steps carried out during the updating work resemble closely Kitchenham's procedure[24], along with some modifications introduced in [20]. However, the process was more difficult to perform than expected, because the update of a SR introduces problems unknown during the first execution. The most relevant problem was to relate the newly obtained empirical results and the previous ones.

Since we had no a glossary of terms from [20], the identification of the treatments (elicitation techniques) and response variables in the 13 new studies and the merge with the first SR's treatments and response variables was a complex task. We realized that the same technique could be named differently in diverse studies, although being the same, because the names are a subjective feature, depending on each author. Likewise, response variables suffered the same problem, aggravated by the fact that not only the name, but also the measurement procedure could vary.

If treatments and response variables of the present and past SR could not be merged, the updating work would be doomed to fail, because the combination of current and past empirical results would be unfeasible. Both SRs would be isolated efforts impossible to relate and, therefore, the number of potential evidences to obtain would be much lower. To solve this problem, it was necessary to catalogue the techniques and response variables tested in the empirical studies analyzed in [20], which implied to read the 26 initial publications besides the other 9 specific of this work. It represented a lot of effort which could be saved if such a glossary would have been constructed during the initial SR.

Apart of this drawback, it was possible to perform the SR with only minor difficulties. The updating process is described elsewhere[25].

3. Main findings

After performing the SR, we obtained more than 60 new empirical results. For reasons of space, those results cannot be shown here but they will be published in [26]. Anyhow, that raw material does not have primary interest for the practitioner. The real interest lies in the combination of those empirical results among themselves, as well as with the results of previous SR. This combination or, more precisely, aggregation process, produces the evidences which can be later used to identify in which situations a

given elicitation technique is useful. For details about how this aggregation process is performed, see [20] as well as [27].

Table 1 shows the evidences obtained after the aggregation process. The evidences shown in the table are only those obtained during the updating work or those obtained in the previous SR but modified by the empirical results newly identified. A comprehensive table can be obtained from [26].

The first column of Table 1 contains the evidence's ID (both for the previous and current SR). This ID is only used to ease the reference to evidences. The second column contains one of the following codes: REFUTES (the newly gathered results refutes a previous evidence), REDUCES (the new results cannot refute a previous evidence, but reduces our confidence in it), REINFORCES (a previous evidence is supported with new compatible results) and NEW (a new evidence, not present in the existing set, has been identified). Finally, the third, fourth and fifth columns are used to specify which studies support the evidence, which are neutral and which ones opposes to it (notice that references 1-9 were the ones analyzed in the present work; the others were analyzed in the first SR).

In some cases this type of table is not adequate to represent all types of evidences and its interpretation may be somehow difficult. For example, evidence 21 does not state a positive fact, such as "transcription time is longer for…", but they say "transcription time CANNOT BE ESTABLISHED AS being longer for…". In experimental terms, it means that no effect has been identified between techniques. In those cases "neutral" studies are really supporting the evidence, and "support" and "opposes" studies deny it. This exception should be considered when reading evidences 12, 16 and 21.

Table 1. Results of the aggregation

ID	KEY	Result of the aggregation	Support	Neutral	Opposes
12	REFUTES	There do not appear to be any differences in terms of session duration between unstructured interviews and laddering *(this evidence is not longer valid)*	[28]	[29,30]	[5]
16	REDUCES	Transcription time cannot be established as being longer for introspective techniques, like protocol analysis, than for unstructured interviews or vice versa	[1]	[28,29]	
21	REINFORCES	Transcription time cannot be established as being longer for sorting techniques than for laddering or vice versa		[1,28,29]	
33	REINFORCES	Laddering gathers fuller information than sorting techniques	[1,28]	[29]	
36	NEW	The efficiency of unstructured interviews is greater than scaling techniques	[5,9]		
37	NEW	Laddering and scaling techniques have the same efficiency	[5,9,28]		
38	NEW	Scaling techniques are more difficult to apply than unstructured interviews	[5,9]		
39	NEW	Laddering is more difficult to apply than unstructured interviews	[5,9]		
40	NEW	Laddering and scaling techniques have the same difficulty	[5,9]		

4. Discussion

The limit of space makes impossible an extensive discussion, so that we will only draw attention to issues related to interviews. The reason is that, in the previous SR, it was found that interviews were overall the most effective elicitation technique, although some contrived techniques like laddering or card sorting were equally effective in some cases[19]. The current work strengthens this conclusion. No empirical result contesting such effectiveness has been identified. Quite the contrary, a competing technique like laddering is found to be more difficult to apply (evidence 39) and therefore can be regarded as less effective than interviews. Concerning efficiency, the same conclusions that in [19] hold. Interviews may be more time-consuming than other more-focused techniques like sorting or protocol analysis. Evidence 16 supports that fact (study [1] suggest that interviews take more time than protocol analysis). Therefore, interviews should be planned carefully to save elicitation time.

5. Conclusions

This paper presents the update of a previous systematic review. This update has the aim of identifying well-founded practices when selecting an elicitation requirements technique. The achieved results show that interviews are in average the most effective elicitation technique. In near future, we want to combine these findings with those from theoretical works and expect opinion to develop a comprehensive theory concerning the application of elicitation techniques.

6. References

References used in the updating work

[1] Burton, A. M., Shadbolt, N. R., Rugg, G., and Hedgecock, A. P., "Knowledge elicitation techniques in classification domains," *Proceedings of the 8th Conference in Artificial Intelligence ECAI-88,* 1988.

[2] Chao, C.-J. and Salvendy, G., "Impact of cognitive abilities of experts on the effectiveness of elicited knowledge," *Behaviour & Information Technology* , vol. 14, pp. 174-182, 1995.

[3] Holsapple, C. W. and Raj, V. S., "An exploratory study of two KA methods," *Expert Systems*, vol. 11, pp. 77-87, 1994.

[4] Wood, L. E., Davis, T. C., Clay, S. L., Ford, J. M., and Lammersen, S., "Evaluation of interviewing methods and mediating representations for knowledge acquisition," *International Journal of Expert Systems*, vol. 8, pp. 1-23, 1995.

[5] Brandt, J. P. and Shook, S. R., "Attribute elicitation: Implications in the research context," *Wood and Fiber Science*, vol. 37, pp. 127-146, 2005.

[6] Bradburn, B. , "A comparison of knowledge elicitation methods," *International Conference on Engineering Design (ICED'91),* pp. 298-305, 1991.

[7] Maiden, N. A. M. and Rugg, G., "Knowledge acquisition techniques for requirements engineering," *Proceedings of the Workshop on Requirements Elicitation for System Specification,* Keele, UK, 1994.

[8] Grabowski, M., "Knowledge acquisition methodologies: Survey and empirical assessment," *Proceedings of the Ninth International Conference on Information Systems,* Minneapolis, MN, USA, pp. 47-54, 1988.

[9] Steenkamp, J.-B. E. M. and Van Trijp, H. C. M., "Attribute elicitation in marketing research: A comparison of three procedures," *Marketing Letters*, vol. 8, pp. 153-165, 1997.

Other references

[10] Boehm, B. W., McClean, R. K., and Urfrig, D. B., "Some experience with automated aids to the design of large-scale reliable software," *IEEE Transactions on Software Engineering*, vol. 1, pp. 125-133, Mar, 1975.

[11] Tavolato, P. and Vincena, K. A Prototyping Methodology and Its Tool. In: *Approaches to Prototyping*, ed. Budde, R. Berlin: Springer Verlag, 1984.

[12] Standish Group, "The CHAOS Report," *http://www.standishgroup.com/sample_research/PDFpages/chaos1994.pdf*, vol. Oct 18, 2005.

[13] SWEBOK. *Software Engineering Body of Knowledge*, http://www.swebok.org, 2005.

[14] Maiden, N. A. M. and Rugg, G., "ACRE: selecting methods for requirements acquisition," *Software Engineering Journal*, vol. 11, pp. 183-192, 1996.

[15] Hickey, A. M. and Davis, A. M., "A unified model of requirements elicitation," *Journal of Management Information Systems*, vol. 20, pp. 65-85, 2004.

[16] Hickey, A. M. and Davis, A. M., "Elicitation technique selection: How do experts do it?," *Proceedings of the Requirements Engineering Conference (RE'03),* pp. 169-178, 2003.

[17] Dyba, T., Kitchenham, B. A., and Jorgensen, M., "Evidence-based software engineering for practitioners," *IEEE Software*, vol. 22, no. 1, pp. 58-65, 2005. ./methods/6.pdf.

[18] Davis, A., Dieste, O., Hickey, A., Juristo, N., and Moreno, A. M., "Effectiveness of Requirements Elicitation Techniques: Empirical Results derived from a Systematic Review," *Proceedings of the IEEE International Conference on Requirements Engineering,* Minneapolis, Minessota, 2006.

[19] Dieste, O., Juristo, J., and Shull, F., "Understanding the Customer: What Do We Know about Requirements Elicitation Techniques?," *IEEE Software*, vol. 25, pp. 11-13, 2008.

[20] Dieste, O. and Juristo, N., "Systematic Review and Aggregation of Empirical Studies on Elicitation Techniques," *IEEE Transactions on Software Engineering*, vol. 2008.

[21] B.A. Kitchenham. *Procedures for performing systematic reviews*, Keele University TR/SE-0401, 2004.

[22] de Bont, C. J. P. M., Consumer Evaluation of Early Product-Concepts 1992. Delft University.

[23] Geiwitz, J. , Kornell, J., and McCloskey, B. P., "An Expert System for the Selection of Knowledge Acquisition Techniques," Anacapa Sciences, Santa Barbara, CA, USA, Technical Report 785-2, 1990.

[24] Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El Emam, K., and Rosenberg, J., "Preliminary guidelines for empirical research in software engineering," *IEEE Transactions on Software Engineering*, vol. 28, pp. 721-734, 2002.

[25] Dieste, O., López, M., and Ramos, F., "Formalizing a systematic review updating process," *Proceedings of SERA'08,* Prague, Czech.

[26] Dieste, O., López, M., and Ramos, F., "Formalizing a systematic review updating process," Universidad Complutense de Madrid, Technical Report, to appear.

[27] Dieste, O."TR5. Agregación de las evidencias obtenidas de los estudios empíricos relevantes," http://grise.upm.es/research_documents.php, 2006.

[28] Burton, A. M., Shadbolt, N. R., Rugg, G., and Hedgecock, A. P., "The efficacy of knowledge acquisition techniques: A comparison across domains and levels of expertise," *Knowledge Acquisition*, vol. 2, pp. 167-178, 1990.

[29] Burton, A. M., Shadbolt, N. R., Hedgecock, A. P., and Rugg, G. A formal evaluation of knowledge elicittion techniques for expert systems: Domain 1. In: *Research and development in expert systems IV: proceedings of Expert Systems '87, the seventh annual Technical Conference of the British Computer Society Specialist Group on Expert Systems*, ed. Moralee, D. S. Cambridge, UK: Cambridge University Press, 1987.

[30] Corbridge, B., Rugg, G., Major, N. P., Shadbolt, N. R. , and Burton, A. M., "Laddering - technique and tool use in knowledge acquisition," *Knowledge Acquisition*, vol. 6, pp. 315-341, 1994.

Automatic Discovery of Interactions Between Software Requirements

Edgar S. Calisaya, Marcos R. S. Borges, Maria Luiza M. Campos
Graduate Program in Informatics, Federal University of Rio de Janeiro, Brazil
edgarsc@posgrad.nce.ufrj.br, mborges@nce.ufrj.br, mluiza@nce.ufrj.br

Abstract

The Requirement Engineering is the first and one of the most critical phases of any software development methodology. This phase is very complex because of the imprecision of the process, the communication problems and the different viewpoints of the stakeholders. Requirements defined with imprecision or ambiguity hide and/or make difficult the discovery of the different interactions that could exist between requirements. These requirements are considered interdependent, as one requirement depends on or affect others. The identification of the interactions between the requirements allows understanding and acting on the impact of these interactions in the subsequent stages of the software development process. There are several methods for requirements specification. However, most of them don't show explicitly these interactions. In this paper, we present an approach that allows the specification, identification and revealing the different interactions between requirements, using a semi-formal method based on events. The events are considered because the flows of events describe the behavior of the system, through a set of interactions between objects.

1. Introduction

Research in requirement engineering has demonstrated that often the set of requirements of a system are not independent. On the contrary, there are different types of interactions among them [1, 2, 3, 4, 5]. This happens because the different elements that compose a system are not isolated entities. On the contrary, the relationships and interactions among these entities make possible the functioning of the system.

The identification of the requirement interactions in an important phase of requirements analysis, supporting the evaluation of the impact of these interactions in the subsequent stages of the process. The knowledge of these interactions allows: to resolve eventual conflicts that could exist, to better plan the implementation of requirements, to manage the impact of a change on other requirements, to trace requirements, and to plan the tests considering the interactions. Several researches show these benefits [1, 7], but most of them don't explicitly show how these interactions influence the subsequent stages of the development process.

Among the approaches for identifying interactions between requirements, we can mention those described by Robinson at al. [4]. However, the main problem of this approach is that it considers a general domain and not only the software domain. Besides, they don't present clearly the type of interaction and why and when they happen.

In the software engineering domain, some approaches for identification of interactions between requirements based in features were proposed [6, 7]. The problem of features interaction is generally understood as a situation where the integration of several features in a system can interfere or affect one another. Shehata [5] and Zhang [7] showed that requirements and features can be matched, as a high level requirement consists of several features, and a feature can be defined as a set of cohesive requirements. The main drawback of these approaches is their limited scope. They fail to consider conflicting interactions, for example. They also depend on detailed information about the domain and implementation-specific knowledge. Besides, if a feature is a collection of cohesive requirements, then it becomes necessary to know and to identify the existent relationships among these requirements.

Among the approaches for detecting requirements interactions described in the literature [4, 5, 6, 7], some are based on informal methods, depending mainly on the designer's experience on the system domain. When formal methods are used, supported by a formal specification language, they are often considered hard to adopt and expensive.

This paper proposes an approach for specifying, detecting and identifying the different interactions between requirements, using a semi-formal method based on events. It brings together most advantages of existent approaches, supporting the identification

of interactions with less effort and complexity: (i) it significantly reduces the user intervention; (ii), it doesn't strongly depend on formal specification languages; (iii) it uses heuristics for interactions detection; (iv) it supports visualization of interactions as graphs and tables; and (v) it allows the discovery of implicit interactions.

The method is based on *events*: the flows of events describe the behavior of the system through a set of interactions between objects, differently from traditional approaches that do not show explicitly these interactions.

This paper is structured as follows. In Section 2 we define and present the types of interactions between requirements considered in our method. Section 3 describes the concepts used in the specification and in subsequent identification of interactions between requirements. In Section 4 we present the proposed method for interactions identification. In Section 5 we show the modules of our framework. Section 6 presents an example of an application of the method. Finally, in Section 7 we give our conclusions so far and the directions for future work.

2. Requirements Interaction

One of the key topics to obtain a set of clearly and precisely defined and specified requirements is how the interactions among them are managed, specially the conflicting or inconsistent relationships.

There is an interaction when two or more requirements have effect on each other. These interactions can be caused by the following reasons: *different view points of several stakeholders, change or re-use of requirements, component-based development, etc*. Some definitions of interactions and the reasons for which these happen can be found in [1, 4].

Among these interactions it is possible to clearly identify two types of general interactions, positive and negative ones. Those of positive type are relationships of intrinsic dependence (Requirement R1 for his realization requires R2), and those of negative type that mainly include conflicting interactions.

2.1. Requirements Interaction Taxonomy

Numerous classifications were generated to represent the types of interactions among requirements, but there isn't a general consensus yet about the best one [1, 3, 5, 6, 7]. The classification presented in the Figure 1 was prepared considering all these classifications. We considered only those types of interactions that are basic and have a significant effect in the remainder of the software development process, especially in the software design.

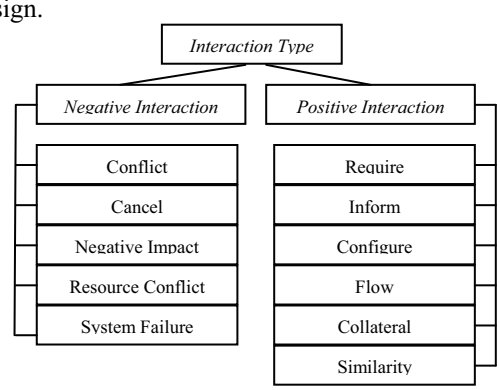


Figure 1. Requirements Interaction Taxonomy

3. Events and Actions Based Requirements Specification

The dynamic models of the Object Oriented Analysis represent the behavior of the system, i.e., the interactions among the different objects of the system and their environment. This interaction among the different objects is caused principally by the presence of some event produced by another object or some external entity. Based on this assumption, it is possible to say that the events are the entities that stimulate and control the functioning or behavior of the system.

The identification of the events is the first stage of the software development process cycle. Particularly in *RE* it allows to know the different interactions that exist among the set of requirements, and subsequently, to know the set of interactions among the different objects or entities of the system.

Some of the reasons why it is important to relate requirements to events and actions they control are:

- The execution of a requirement produces a result (event);
- The events cause the execution of functionalities or the creation of an object;
- The objects interact through the events.

The knowledge about the events and the actions involved in each one of the software requirements, in the initial stages of the software development process, also allows having a better traceability of the requirements implementation in subsequent stages. El-Ansary [9] presents some reasons why the modeling or development of systems should be directed by *Events*.

3.1. Basic Attributes of Requirements

After all requirements are described and listed textually, they should be decomposed and represented in the form of a sequence of attributes that consists of events, actions, states and resources. See Table 1.

Some of the concepts presented in this section were extracted of the Object Oriented Dynamic Modeling: A Comparative Analysis of Techniques [8], the Business Process Modeling Notation Specification adopted by the OMG [10] and [5, 9].

Table 1. Attributes of Requirements

Attribute	*Description*
IDR	The identifier of the requirement.
Description	The description of the requirement
Event	They are the incidents or facts happened inside or out of an object.
Action	They are the activities carried out during the execution of the requirement, such as calculations, *generation of events*, etc.
Object	The objects involved in the execution of the requirement.
Resource	Instruments or tools used by the requirement to complete his execution.

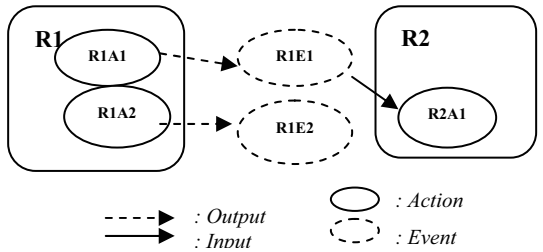


Figure 2. The Role of the Events

3.2. Basic Attributes of Events

To investigate the possible roles of the events in the set of interactions among requirements, it is necessary to understand the definition and the description of each one of them. Based on this observation, we identify the need for specifying an event. For this, each requirement has a set of associate events, and it is necessary to specify each one of them. See Table 2.

Figure 2 illustrates the role of the events and actions in the interaction among the requirements. The Requirement R1 executes two actions: R1A1 and R1A2; R1A1 produces (output) the event R1E1 and R1A2 produces the event R1E2. In another side the Requirement R2 executes the action R2A1, R2A1 is stimulated by R1E1 (input).

Table 2. Attributes of Events

Attribute	*Description*
IDE	The identifier of the event.
Description	The description of the event.
IDR	The identifier of the requirement.
Type	*Message, Time, Rule, Link, Multiple and Cancel* [10].
Category	▪ *Input: it stimulate some action.* ▪ *Output: it's generated by some action.*
Action	The Action that *produces it* or is *stimulated* by the *event*.
Object	The event causes changes of state of objects. ▪ *Pre-state.* ▪ *Next-state.*
Resource	Resources stimulated by the *event*.

4. Discovering Requirements Interactions

For each one of the interactions types shown in Figure 1, a set of rules for interaction detection was defined and created. These rules involve each of the attributes defined in Tables 1 and 2. All the rules identified were built based on the *template*:

WHEN <*Event*>
[**IF** <*Pre-condition* >]; it matches the object states, events, actions and resources of two requirements.
THEN <*Interaction Type*>

In the Tables 3 and 4, we present some of the rules identified for some of the types of interactions.

To facilitate and to reduce significantly the number of comparisons to match each one of the requirements, it can be necessary to identify the common events to each one of them. See Table 5.

Table 3. Cancel Interaction.

Interaction Type	Cancel.
Description	When the execution of *Rx* overrides or cancels the execution of *Ry*. ▪ The event produced in *Rx* cancel the action executed en *Ry*. ▪ Requirements: *Rx* and *Ry*; *Rx != Ry.*
Rule	**WHEN** Event **IF** ***Rx***.Event = Event **AND** ***Rx***.Event != ***Ry***.Event **AND** ***Rx***.Event.Type = *Cancel* **AND** ***Rx***.Event.Category = In*put* **AND** ***Rx***.Event.Action = ***Ry***.Event.Action **AND** ***Rx***.Event.Object = ***Ry***.Event.Object **THEN** ***Rx*** *Cancel* ***Ry***

Table 4. Conflict Interaction

Interaction Type	Conflict.
Description	Set *Rx* and *Ry*, there is a condition *B* that can cause a conflict. ▪ *Rx* and *Ry* are stimulated by the same Event and share the same Object (different next-state) and Action. ▪ Requirements: *Rx* and *Ry*; *Rx != Ry*.
Rule	**WHEN** Event **IF** ***Rx***.Event = Event **AND** ***Rx***.Event = ***Ry***.Event **AND** ***Rx***.Event.Object = ***Ry***.Event.Object **AND** ***Rx***.Event. Action = ***Ry***.Event. Action **AND** ***Rx***.Event.Object.Pre-State = ***Ry***.Event.Object.Pre-State **AND** ***Rx***.Event.Object.Next-State != ***Ry***.Event.Object.Next-State **THEN** ***Rx*** *Conflict* ***Ry***

Table 5. Listing Events

IDE	*Description*	*Common Requirements*
The identifier of the event.	The description of the event.	The requirements related to this event.

5. Architecture

In this section we describe the functioning of the method, the architecture of our framework (Figure 3) and each one of its components. Our framework consists of several modules that store and process different types of information to determine and to detect the existent interactions among the set of requirements presented to the framework. In what follows, each one of these modules are briefly described:

- *Requirements*: Initially the requirements are listed and described textually.
- *Requirements Repository*: After the requirements are listed and decomposed in more simple, are stored in the repository.
- *Discovering Interaction*: After the requirements are stored, a requirements analyst begins the process of identification of attributes for each one of them. Then it should identify the attributes of the events associated with them. After that, the detection of the interactions process (automatically) is carried out using the set of detection rules stored in the *Interaction Taxonomy* module.
- *Interaction Taxonomy:* This module stores the taxonomy of the types of interactions (Figure 1) and the rules of detection of interaction. Such as depicted in Tables 3 and 4.
- *Results*: This module allows display and validates the graphs and tables of interaction.
- *Requirements Specification*: This module stores the set of specified requirements based on the attributes listed in the Tables 1 and 2.

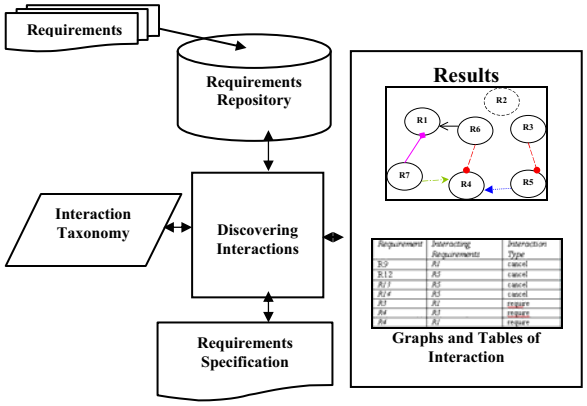


Figure 3. Architecture

6. Case Study

We have been evaluating the effectiveness of our method in several domains. To illustrate the method we show a case study using the Lift Control System. In this case study, a set of 14 requirements (Table 6) describes the basic operation of a simple Lift. A detailed and complete description of this case study can be found in [5].

The Lift is composed of:

- Call Button in each floor.
- Open-Door Button inside the Lift.
- Buttons for each floor inside the lift.

Table 6. The Lift Control System Requirements

IDR	*Description*
R1	The lift is called by pressing a call button, either at a floor or inside the lift.
R2	Pressing a call button is possible at any time.
R3	When the lift passes by floor K, and there is a call for this floor, then the lift will stop at floor K.
R4	When the lift has stopped, it will open the doors.
R5	When the lift doors have been opened, they will close automatically after d time-units.
…	…
R12	The closing of a door may be prevented by pressing the open-door button.
R13	When something blocks the door, the lift interrupts the process of closing the door and reopens the doors.
R14	When the lift is overloaded, the door will not close.

Table 7. Identifying Attributes for the Requirements

IDR	*Event*	*Action*	*Object*	*Resource*
R1	Pressing the call button.	Call the lift.	The lift. The Call Button (in/out).	
R2	Pressing the call button.		The Call Button (in/out).	
…	…	…	…	…
R5	The doors are opened. The doors are closed.	Close the Doors automatically after *d* time units.	The doors. The time counter.	
…	…	…	…	…
R13	Something blocks the doors. The doors are opened.	Close the Doors automatically after *d* time units.	The lift. The doors.	The block sensor.
R14	The lift is overloaded. The doors are opened.	Close the Doors automatically after *d* time units.	The lift. The doors.	The Overload sensor.

After the requirements are listed, it proceeds to the identification of the attributes (See the Tables 7). The Table 8 presents the common events to the requirements.

Table 8. Identifying Events

IDE	*Description*	*Common Requirements*
E1	Pressing the call button.	R1,R2
E2	The lift is called from the floor K.	R1, R3, R9, R10
E3	The lift passing through the floor K.	R3
E4	The lift is stopped.	R3, R4, R7, R9, R10
E5	The doors are opened.	R4, R5, R9, R10, R12, R13, R14
…	…	...
E12	Something blocks the doors.	R13
E13	The lift is overloaded.	R14

The Tables 9 and 10 show the detailed attributes of the events *E5* and *E13* listed in the Table 8.

Table 9. Identifying Attributes for the Event E5 and the Requirement R5

Attribute	*Description*
IDE	E5
IDR	R5
Type	*Link*
Category	*Input*
Action	Close the Doors automatically after d time units.
Object	▪ *Pre-state: The Doors are opened.* ▪ *Next-state: The Doors are closed.*

Table 10. Identifying Attributes for the Event E13 and the Requirement R14

Attribute	*Description*
IDE	E13
IDR	R14
Type	*Cancel*
Category	*Input*
Action	Close the Doors automatically after d time units.
Object	▪ *Pre-state:* The Doors are opened. ▪ *Next-state:* The Doors are opened.

When after the previous activities were carried out, it proceeds *Identifying the Interactions between the Requirements*.

- *Interaction Type:* Cancel.
- *Requirements:* R14 -> R5.
- *Event:* E13 -> The Lift is overloaded.
- *Action:* Close the Doors automatically after d time units.
- *Object:* The Doors.

When applying the rule of the Table 3, it is possible to infer that the event of the Requirement R14 (E13: The Lift is overloaded.) cancels the execution of the action implemented in R5 (Close the Doors automatically after d time units).

Table 11. The Identified Interactions

Requirement	*Interacting Requirements*	*Interaction Type*
R9	R1	cancel
R12	R5	cancel
R13	R5	cancel
R14	R5	cancel
R3	R1	require
R4	R3	require
R10	R1	require

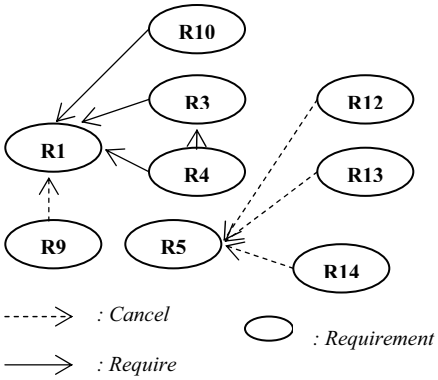


Figure 4. Interactions Graph

The Table 11 and the Figure 4 show all the identified interactions applying the method.

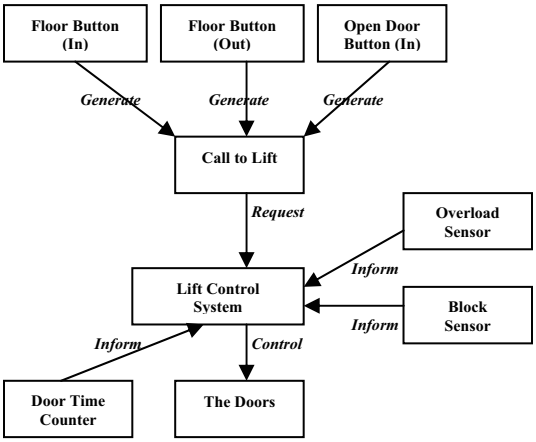


Figure 5. Initial Class Diagram

7. Conclusions

In this paper, we have presented our approach to specify, detect, and automatically discover the interaction types among the software requirements. The approach is based on events and actions and uses a semi-formal method. The method requires neither the user intervention nor any external knowledge for the *identification of interactions*. The method reduces the user intervention, because the identification of interactions is done using the defined rules of interaction detection, such as Table 3 and 4. The method is able to identify most interaction types described in the literature (negative and positive interactions).

The method proposed in this paper works well if a requirements analyst can properly identify requirements attributes like in Table 7; however this work is less complex than specify the requirements using a formal specification language.

In the future, we will expand the method to cover additional interaction types, which were not considered in this paper. Moreover, we intend to work on the improvement of the method by identifying and improving the rules of interaction detection. This will enable us to make the algorithms more efficient and precise.

We are also working on a software prototype to support the user to validate, improve or reject the interactions identified by our method.

References

[1] Å.G. Dahlstedt, and A. Persson, Requirements Interdependencies - Moulding the State of Research into a Reseach Agenda, In *Proceeding of the Ninth International Workshop on Requirements Engineering: Foundation for Software Quality*, Austria, 2003, pp. 71-80.

[2] J. Giesen, and A. Volker, Requirements Interdependencies and Stakeholders Preferences, In *Proceedings of IEEE Joint International Conference on Requirements Engineering*, 2002, pp. 206-209.

[3] X. F. Liu, and J. Yen, An Analytic Framework for Specifying and Analyzing Imprecise Requirements, In *Proceedings of International Conference of Software Engineering,* 1996.

[4] W. N. Robinson, S. D. Pawlowski, and V. Volkov, Requirements interaction management, *In ACM Computing Surveys 35 (2)*, 2003, pp. 132-190.

[5] M. Shehata, Detecting Requirements Interactions using Semi-Formal Methods. PhD. Thesis (Doctor of Philosophy), Department of Electrical and Computer Engineering, Calgary University, Alberta, Canada, 2005.

[6] L. Yuqin, Y. Chuanyao, Z. Chongxiang, and Z. Wenyun, An Approach to Managing Feature Dependencies for Product Releasing in Software Product Lines, *In Proceedings of the International Conference on Software Reuse – ICSR, LNCS 4039*, 2006, pp. 127-141.

[7] W. Zhang, H. Mei, and H. Zhao, Feature-driven Requirement Dependency Analysis and High-level Software Design, *Requirements Engineering Journal*, volume 11, number 3, 2006, pp. 205-220.

[8] G. Bustos, and C.A. Heuser, Object Oriented Dynamic Modeling: A Comparative Analysis of Techniques, In *Brazilian Symposium of Software Engineering,* Recife, Brazil, 1995.

[9] A. El-Ansary, Behavioral Pattern Analysis: towards a new representation of systems requirements based on actions and events, In T*he ACM Symposium on Applied Computing,* Madrid, Spain. 2002.

[10] BPMN, Business Process Modeling Notation, OMG, 2006.

A Model-Driven Approach for Software Product Lines Requirements Engineering

Mauricio Alférez, Uirá Kulesza, André Sousa, João Santos,
Ana Moreira, João Araújo, Vasco Amaral
Dept. Informática, FCT, Universidade Nova de Lisboa, Portugal
{mauricio.alferez, uira, als, jps, amm, ja, vasco.amaral}@di.fct.unl.pt

Abstract

UML and feature models complement each other well and can be the base techniques for a systematic method to identify and model software product line (SPL) requirements. In this paper, we present a model-driven approach to trace both features and UML requirements analysis model elements, and to automatically derive valuable models for domain and application engineering. The resulting contribution is a synergetic approach for SPL requirements. We illustrate it by using a home automation system product line.

1. Introduction

Software product line (SPL) approaches [1-3] aim at improving the productivity and quality of software development by enabling the management of common and variable features of a system family. A system family is defined as a set of programs that shares common functionalities and maintains specific functionalities that vary according to specific family product members. A SPL can be seen as a system family that addresses a specific market segment [1].

Over the past few years, several SPL development approaches have been proposed [1-4]. Most of them motivate the identification of common and variable features of the SPL by means of domain analysis activities. A feature [4] can be seen as a system property or functionality that is relevant to some stakeholder and is used to capture commonalities or discriminate among products in a SPL. SPL features are typically represented in domain analysis using feature models [5]. Other requirements models (e.g., use case and activity models) can be used to better describe and detail the SPL requirements. The feature and requirements models are then used as a reference along all the process to guide the development of the SPL.

Some research works have addressed the use of feature models in combination with other models. Approaches like [6] and [7] propose to create relationships between features and UML models by means of intrusive graphical elements such as, presence conditions or notes to indicate variability. The main disadvantage of these approaches is the creation of convoluted and polluted models, which bring difficulties to understand, maintain and scale the models and trace links between features and UML elements.

Other approaches [3, 8, 9] give some directions on how to model and trace variability information. However, and similarly to what happens with the previous approaches, they do not provide specific activities and tool support for modeling, tracing and generating requirements models for specific products based on the tracing information.

This paper presents a model-driven approach for variability management in product lines that addresses traceability between features and UML requirements models (like use cases and activity models). The main contribution is to show how model-driven techniques can be used to automatically derive, from the information provided by the trace links, requirements models for specific products of a SPL, and views that explicitly illustrate the relationships between features and UML requirements model elements. These views are useful in both domain and application engineering stages. The general idea of our approach is to apply bindings between metamodels, create a simple tracing metamodel strategy, generate specific product requirements models automatically, and use composition rules to specify compositions between use cases by means of their respective activity diagrams.

This paper starts with an overview of our metamodelling strategy and approach main activities in

Section 2. These activities are illustrated using a home automation system, in Section 3. Section 4 explores and presents lessons learned from the application of our approach. Finally, Section 5 concludes the paper and points out directions to some future work.

2. A Model-Driven Approach for SPL Requirements Engineering

Traceability between feature and requirements models is supported in our approach by a metamodelling strategy. Figure 1(a) introduces the adopted metamodelling strategy and Figure 1(b) makes that strategy concrete through feature, use case and activity metamodels.

A variability model is used to represent the common and variable SPL features. One or more requirements models detail the SPL requirements. A traceability metamodel is used to link abstractions from the variability and the requirements models. This enables the navigation across abstractions of the different types of models using model-driven techniques and tools. The traceability model also supports backward and forward traceability between a feature model (or any of its configurations) and requirements models. Each configuration defines the features of a specific product from the SPL.

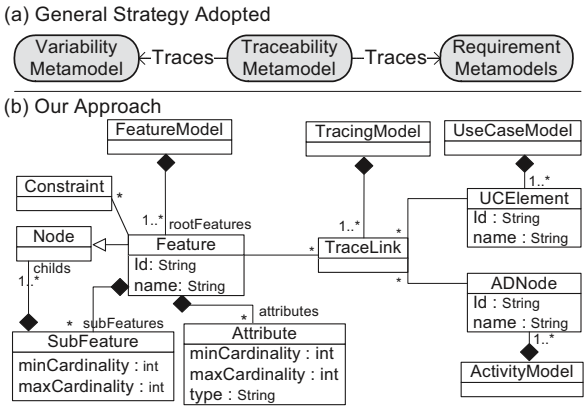


Figure 1. Traceability support strategy

Our approach adopts a feature metamodel based on [7] as the variability model. UML use case and activity models specify the SPL requirements. Due to the large dimension of the their metamodels, we only show the use case model element "UCElement" and activity diagram node "ADNode" from which all the traceable elements of each model can be inherited. Activity diagrams model the behavior of use cases. Use cases and activity models are related to each other by means of the feature to which they are connected.

Our metamodel (Figure 1(b)) supports the set of models that we create in domain and application engineering. The metamodel enables the creation of models in *conformance to* their respective metamodels [10], and helps to understand the relationships between the models elements. Besides the metamodelling strategy, our approach also defines a set of systematic activities in the domain and application engineering stages. The SPL requirements models are created and manipulated during these stages using model-driven techniques and tools.

At the domain analysis level, we perform the activities described next. Although they are organized sequentially, they are typically executed iteratively and incrementally.

1. Identify requirements. The SPL requirements can be elicited using traditional requirements engineering techniques such as inspection of existing documents that describe the problem domain, existing catalogues [11], stakeholders interview transcripts or by using mining techniques [12]. Other approaches such as [9] and [3] already address this activity in detail.

2. Group requirements into features. During this activity, we organize the SPL requirements into clusters according to the specific SPL features they are related to. There are semi-automatic clustering techniques such as [13] that could help to support this activity. However, the specific steps followed in the clustering sub-process are out of the scope of this paper and are not included due to lack of space.

3. Refactor requirements and features. During the previous activity, requirements could result to be linked to more than one feature. We propose to refactor those requirements to be ideally related to only one feature, whenever possible. It contributes to achieve a better modularization of the SPL requirements through the separation of the variable parts of each requirement [14] as well as facilitate establishing tracing links between requirements and features.

4. Model SPL features and use cases. This activity structures and represents the SPL requirements using use case and feature models. Use case models specify the functional requirements and feature models specify the SPL features and variability-commonality information.

5. Relate features to use cases. The relationships between features and use cases are specified visually in a table of trace links. The table allows defining and maintaining the trace relationships between features and UML elements.

6. Generate SPL use cases annotated with features. A model-driven tool developed for our approach uses the relationships between use cases and features to generate specific use case models annotated with features [15]. In the annotated model, each use case is shown with the respective(s) feature(s) related to it. Therefore, it is also possible to obtain the set of use cases related to a specific feature. This allows the domain analysis engineers and SPL architects to reason about how each use case is related to the SPL features and to analyze the impact of change of specific features in SPL requirements.

7. Model use cases as activity diagrams. The detailed behavior of each use case is modeled using activity diagrams, similarly to what happens in several UML-based methods, such as RUP [16]. Use cases specified as activity diagrams, in contrast with textual-based specifications, allows us to enable the use of model-driven generation tools by providing models that conform to a metamodel (i.e., UML activity diagram metamodel) and to help to avoid ambiguity in the specifications [3]. The detailed specification of use cases as activity models also enables us to customize the behavior of use cases according to the features selected to a specific SPL configuration.

8. Specify composition rules between use cases. Each composition rule defines how a variable use case (i.e., linked to a variable feature) can interfere or modify the normal execution of a mandatory use case (i.e., linked to a common feature). Composition rules are defined in terms of the elements of the activity diagrams (e.g., activities, initial state or final state).

The models produced during domain engineering are used in application engineering to generate use case and activity models for specific SPL configurations. We define three activities in application engineering:

1. Define a SPL configuration. The application engineer specifies a SPL configuration, where s/he chooses which optional and alternative features are going to be part of the final application.

2. Generate a use case model from a SPL configuration. Our tool [15] generates the use case model related to the SPL configuration defined in the previous activity. The input for the generation is the SPL use case model, the SPL configuration and the table that maintains the trace links between features and use cases.

3. Generate activity diagrams from a SPL configuration. Our tool is also used to generate activity diagrams related to a specific SPL configuration. In this process, the original activity diagrams can be composed using the composition rules defined in the domain engineering stage. The choice of which composition rules will be used is based on the features included in the SPL configuration. The activity diagram of each extension use case, for example, can be composed with mandatory use cases if the variable feature related to it was selected by the application engineer (step 1 of application engineering).

3. Applying the Approach to a Case Study

To illustrate the activities described in the previous section, we have chosen a home automation system, called Smart Home (see also [3]). This system is one of the SPL case studies proposed by the industrial partners of the European project AMPLE [17]; due to its complexity, we will focus only on a subset of the Security module.

The requirements and feature identification, and refactoring activities, are described in [18]. They provided the features and requirements of our case study. By inspecting those requirements and features, we modeled the SPL feature and use case models. Figure 2 shows the most relevant artifacts produced by the activities of our approach. It shows how each artifact produced in the domain engineering perspective is used to create or derive other artifacts for a specific product in the application engineering perspective. Next we describe the domain engineering activities from our approach.

Model SPL features and use cases. Figure 2(a) shows the feature model of our Security module. It has three main features: *Room Surveillance*, *Admittance Control* and *Intrusion Detection*. *Room Surveillance* is an optional feature that includes *Indoor Camera Surveillance* and, optionally, *Indoor Motion Detection*. The inhabitant can be admitted to enter the house after passing either a *Biometrical Analysis*, *Smart Card*, or entering a *PIN*. In case of selecting intrusion detection, the *Glass Break Detection* must be included and optionally, motion detection sensors and cameras for outdoor security. The notation used in Figure 2(a) is described in Figure 2(j).

We can obtain the SPL use cases from the requirements and features previously identified. Use case modeling is used to better structure the SPL requirements and add more semantics to the features [6]. Figure 2(b) shows the use case model of the case study. The initial SPL features and use cases can be refined and incremented to consider new variabilities or products that need to be included in the family. Both use case and feature models must be updated when

new features are considered or existing ones need to be modified or removed.

Relate features to use cases and generate SPL use case models annotated with features. So that traceability can be maintained between use cases and features, we define an activity to specify the trace relationships between those artifacts. By inspecting the requirements and features of the case study, we related, for example, the *Open Front Door* use case with *Admittance Control*, refined into *Biometrical Analysis*, *Smart Card*, and *PIN* features because to open the front door, the system requires *Admittance Control.* Figure 2(c) shows one of the views that can be generated using the trace links relationships between use cases and features. The open branch in the tree-like structure shows that the *Smart Card* feature is related with the use cases *Identify User by Smart Card*, *Open Front Door* and *Configure Security Management*.

The traceability views of the relationships between features and other artifacts allow the domain analysis engineers and SPL architects to reason about the domain analysis artifacts interdependencies. Currently, there are two kinds of traceability views that our approach can generate in this activity: (i) A use case model annotated with the respective related features; and (ii) a tree structure that shows the list of use cases with the related features and, optionally, the list of features with the related use cases (as in Figure 2(c)).

Create activity diagrams. The behavior of each use case can be specified in our approach using activity diagrams. These diagrams were created by inspecting the requirements. Figure 2(e) and Figure 2(f) shows, for example, the activity diagrams of the *Identify User by Smart Card* and *Identify User* use cases.

Specify composition rules between use cases. Use cases composition is addressed in our approach by means of a set of composition rules. Each composition rule defines how a use case can interfere, modify, or replace the execution of another use case. The composition rules are defined in terms of activity diagrams elements (i.e., activities, initial and final nodes). Composition rules are used during the application engineering phase to derive the specific behavior of use cases for a SPL configuration or product. Figure 2(d) presents the composition rule between the use cases *Identify User* and *Identify User by Smart Card*. It shows how the *Identify User by SmartCard* use case can modify the *Identify User* use case to include additional steps related to the *Smart Card* variable feature. The application of the composition rule is shown in the following subsection where specific activity diagrams can be generated for each product of the SPL.

Next, we describe the execution of the application engineering activities of our approach in the context of the Smart Home case study.

Define a SPL configuration and generate the related use cases and activity diagrams. The first activity in application engineering is to specify a SPL configuration to decide which features will be part of the final application. Figure 2(g) shows a configuration of the case study feature model shown in Figure 2(a) (see the notation used in Figure 2(j)).

Based on the feature model configuration, the relationships between use cases and features, and the SPL use case model (Figure 2(b)), a use case model can be automatically derived using the tool from our approach [15]. Figure 2(h) shows the use case model of the product specified in Figure 2(g).

The final activity of application engineering in our approach involves the automatic customization of the activity diagrams related to each of the SPL use cases using the composition rules specified in domain engineering. Only the activity diagrams of the use cases that are part of the SPL configuration are customized. Figure 2(i) shows the composition between the activity diagrams that describe the *Identify User* and *Identify User by Smart Card* use cases depicted in Figure 2(f) and Figure 2(e), using a `Replace with` composition rule depicted in Figure 2(d). It is not the aim of this paper to present a full-fledged composition language; we just show how it would look like. A complete composition language is one of our aims for future work. For additional details about the current version of our composition language, please refer to [17].

4. Benefits and Lessons Learned

In the context of the European project AMPLE, experiments with our approach have shown that the information of the relationships among the SPL requirements models can be used to support: (i) forward and backward traceability between features and requirements models like use case and activity models; and (ii) reasoning about the impact of feature interactions in the SPL requirements (expressed by the use cases and activity diagrams). Forward and backward traceability enables the creation of tracing queries over all the requirements artifacts and the derivation of specific requirements models for a determined product in the SPL using an model-driven derivation tool, as the one that we have developed [15]. In addition, it enables to the developers to visualize the features changes effects in the SPL requirements through the automatic modification of the

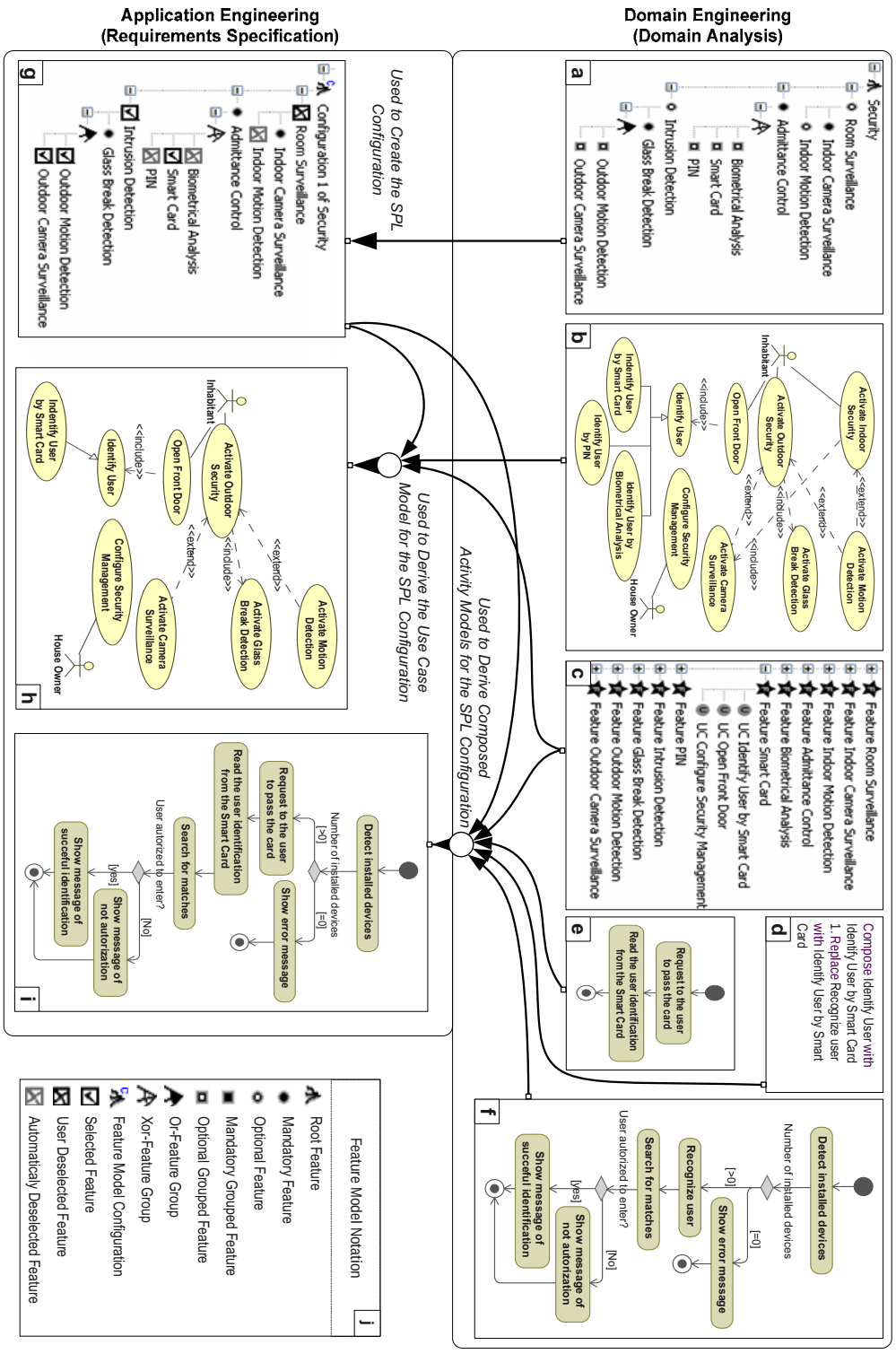


Figure 2. Some of the artifacts produced in the Smart Home Security module case study. (a) SPL feature model; (b) SPL use case model; (c) Use cases related to features; (d) Composition rule between "Identify User" and "Identify User by Smart Card"; (e) Activity diagram of the "Identify User by Smart Card" use case; (f) Activity diagram of the "Identify User" use case; (g) Configuration of the SPL feature model; (h) Use case model for a specific product; (i) Composing "Identify User" with "Identify User by Smart Card"; (j) Feature model notation.

models. On the other hand, the information about feature interactions offered by our approach is useful during the design of SPL architectures to allow an adequate modularization and implementation of their respective features. However, in this paper we have only concentrated on describing the traceability functionalities.

Our metamodelling strategy (Section 2) brings the following benefits to the definition of our approach: (i) *simplicity* – the integration between the metamodels of the feature and requirements models is very easy to understand and evolve; and (ii) *flexibility* – the strategy can be applied to any requirements notation that has a well-defined metamodel.

Our approach also enables composition of crosscutting use cases by representing their steps in activity diagrams. Composition rules are used to specify how the behavior of a use case can affect the behavior of another one. We believe this is an effective way to represent how the SPL variabilities occur along the use cases behavior. The integrated use of these activity diagrams, composition rules and a SPL configuration allows generating the specific behavior of a SPL product. The resulting activity diagram representing the use cases of a product can then be used with different purposes, such as, for example, to document the final requirements of the product or to generate specific test cases for the product.

5. Conclusions and Future Work

This paper presented a model-driven approach to model, specify and trace SPL features and requirements supported by an automated tool. We adopted a simple but useful metamodel integration strategy to allow tracing between features and other requirements models. The approach includes domain and application engineering activities, both illustrated using the Smart Home SPL case study.

Our work is currently being extended to address additional perspectives, such as: (i) to provide more explicit guidance for non-functional requirements and feature interactions modeling and to create special trace views for these concerns; (ii) to deal with uncertainty or volatile requirements in SPLs; (iii) to continue exploiting the activity diagrams to model scenarios [19]; and (iv) define a more complete approach in the context of the AMPLE project to provide tracing support from features and requirements models to artifacts of later software development stages, such as, architecture models and source code. Finally, a full-fledged composition language will be defined.

Acknowledgements. The authors are partially supported by EU Grant IST-33710: Aspect-Oriented, Model-Driven Product Line Engineering (AMPLE).

References

[1] P. Clements and L. M. Northrop, *Software Product Lines: Practices and Patterns*. Boston, USA: Addison-Wesley, 2002.
[2] D. M. Weiss and C. T. R. Lai, *Software Product-line Engineering: a Family-based Software Development Process*. Boston, USA: Addison-Wesley, 1999.
[3] K. Pohl, et al, *Software Product Line Engineering: Foundations, Principles and Techniques*. Berlin, Germany: Springer, 2005.
[4] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*: ACM Press/Addison-Wesley, 2000.
[5] K. Kang, et al, "Feature-Oriented Domain Analysis (FODA) Feasibility Study", SEI, CMU/SEI-90-TR-021, 1990.
[6] K. Czarnecki and M. Antkiewicz, "Mapping Features to Models: A Template Approach Based on Superimposed Variants", presented at GPCE, Tallinn, Estonia, 2005.
[7] A. Bragança and R. J. Machado, "Automating Mappings between Use Case Diagrams and Feature Models for Software Product Lines", presented at SPLC, Kyoto, Japan, 2007.
[8] H. Gomaa, *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*: Addison-Wesley, 2004.
[9] M. L. Griss, et al, "Integrating Feature Modeling with the RSEB", presented at ICSR, 1998.
[10] J. Bézivin, "On the Unification Power of Models", *Software and Systems Modeling*, vol. 4(2), pp. 171-188, 2005.
[11] L. Chung, et al, *Non-Functional Requirements in Software Engineering*, 1 ed: Kluwer Academic Publishers, 1999.
[12] A. Sampaio, et al "EA-Miner: A Tool for Automating Aspect-Oriented Requirements Identification", in *Proceedings of ASE*, Long Beach, CA, USA, ACM Press, 2005, pp. 352-355.
[13] K. Chen, et al, "An Approach to Constructing Feature Models Based on Requirements Clustering", in *Proceedings of RE*, Paris, France, IEEE Computer Society, 2005, pp. 31-40.
[14] A. Moreira, J. Araújo, and J. Whittle, "Modeling Volatile Concerns as Aspects", presented at CAiSE, Luxemburg, Luxemburg, 2006.
[15] "AMPLE Project Research Group at FCT/UNL", http://ample.di.fct.unl.pt/.
[16] P. Kruchten, *The Rational Unified Process: An Introduction*: Addison-Wesley, 2003.
[17] AMPLE, "Ample Project", http://www.ample-project.net/.
[18] M. Alférez, et al, "Traceability between Features and UML-Based Requirements Models: A Model-Driven Approach for Product Lines Engineering", http://ample.di.fct.unl.pt/tool/Alferez-etal-TR-1-2008.pdf
[19] N. Maiden and I. Alexander, *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*: John Wiley & Sons, 2004.

Model Interpretation for Executable Observation Specifications

Mathias Funk, Piet van der Putten, Henk Corporaal
Dept. of Electrical Engineering, Electronic Systems Group
Eindhoven University of Technology, The Netherlands

E-mail: {m.funk, p.h.a.v.d.putten, h.corporaal}@tue.nl

Abstract

Observation functionality integrated into interactive products can help companies identifying current consumer requirements and expectations. As these needs can change rapidly, detailed information about product usage that comes from habitual interaction is crucial to evaluate product acceptance and relevance. We explore how products can be extended with observation functionality that satisfies the information needs of multi-disciplinary experts in the development team. In the process of product evaluation, information requirements are bound to change, and so is the observation behavior. Our approach addresses this by integrating observation functionality into products which can be adapted to current information needs. This paper presents a novel way to remotely configure products in the field by using high-level models, graphical observation specifications, that are interpreted by a runtime environment built into the products in the field. An industrial case-study shows the applicability of the approach. This work is part of ongoing development aiming at a generic observation integration methodology.

1. Introduction

Complex consumer electronics products as well as other innovative product categories nowadays integrate many different features in order to serve a large group of customers. The mass of functions has to be accessed through a user interface which in turn gets more and more complicated. Users have problems to find their ways. Increasing numbers of returned products without any detectable failures suggest this [3].

Furthermore nowadays product creation processes are characterized by high complexity of products and they are influenced by rapidly changing customer demands. Hence, an up-front specification of the product becomes hard if not impossible. In the past, products could be improved in the next version, but today the markets often demand completely new products. Technologies have to reach a level of maturity in a shorter time. The lack of information about the customers' needs leads to a situation where companies press functionality into products, thus entering a vicious cycle of complexity [1]. This blurs the customer's understanding of the product and a match between customer expectations and the actual capabilities of the product becomes even more unlikely.

An approach to address this industry-wide problem is to get representative user feedback on try-out products or prototypes [2]. Traditionally this is done by collecting customer opinions in questionnaires and video-taping user interactions with the product in usability labs. Nowadays, with almost ubiquitous internet access, other methods can be used which are expected to provide much richer data on the actual used product features and user preferences. The integration of observation modules into products can enable data collection according to the actual and ever changing information needs of the product development team.

Our research aims at the introduction of observation integration or *design for observation* as a first class development task, because the delivery of relevant information of use and possibly user expecations will become more important in the future of product development.

In this work we address a problem that occurs when the development processes are not yet tuned to observation integration in an efficient way: observation is brought into products late in the development process. Due to changing requirements for observed information, the implemented observation functions have to be adapted regularly. This holds especially for the use phase, when products are given to testers. Then, highly adaptable and remote observation is crucial. This causes a substancial effort not only for developers but also for observation specification and certainly for the alignment of both. If the adaptation mechanisms are not automated, product evaluation becomes difficult if not impossible. Therefore automation of observation specification deployment is a primary precondition for such product evaluation methods.

More precisely, we address the technical transition process between observation specification and the dynamic execution of a translated specification, possibly to be constructed during runtime. As a result, observation facilities support this scenario of remotely adapted observation via a communication channel like the internet. Figure 1 shows an overview on such a system, consisting of an authoring environment, a server instance, several products in use and further services dealing with the analysis and visualization of collected product usage information. We refer to [8] for an explanation of system and its use to model users and their interaction with a consumer electronics product.

In the following sections, after pointing at related work, the approach of observation modeling is explained together with a description of visual specification and observation modules. This is followed by the core section about model interpretation. The paper ends by presenting a case study which shows the applicability of the approach in the context of new product development.

2. Related work

The research on observable products as described in this paper is on the one hand strongly connected to the field of user modeling. The framework we developed stands in the tradition of generic user modeling systems (for an extensive overview see [12]). On the other hand, it is in the area of remote runtime monitoring where there have been efforts to monitor deployed software [4] and to use logging inside products [13]. The use of a client-server architecture for information distribution across a network of products is straight-forward in this domain and has been described before [10]. However, our approach of remotely changeable observation behavior differs from traditional monitoring as we do not assume a pre-defined set of information sources, but deal with constantly changing information requirements. In this sense, the research stands also in relation to adaptive software [11]. This paper tackles the problem of *flexible instrumentation of observation modules*. Our approach is based on the specification of observation by use of a domain-specific language [6]. The specified observation is executed on products, which is an application of model interpretation [5]. Compared to well-known model-driven approaches like MDA and MDE [14, 7] this technique offers a dynamic transformation shortcut from model to executable.

3. Observation

The observation of remote systems potentially covers a wide range of complex electronic products. It is seldomly done in an engineering approach aiming at reuse and a long-term application. Especially for product families observation gains importance: it is one of the few system parts which are easiest to generalize. Moreover, the collected information has a large influence on the specification and the targeting of future products within the product family.

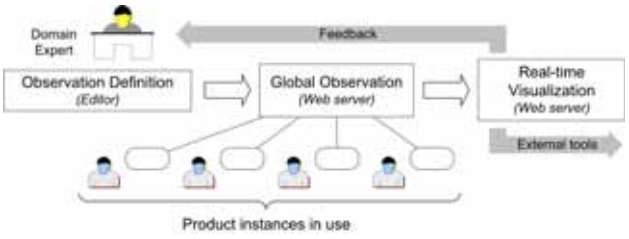


Figure 1. Framework overview

Observation is done in several subsequent steps: Information is sensed by so called *hooks* which might imply that an information source is either triggered periodically for data or raises an event itself. Resulting low-level data is processed in the next step and can herewith be aggregated, normalized or temporarily cached. This preprocessing stage yields complex events which result from the combination of multiple low-level sources. Depending on the extent of aggregation and event correlation those events can carry enough semantic information to be relevant for analysis by information stakeholders. Finally, the data has to be collected centrally which allows for real-time visualization and post-processing using external tools. Obviously, information capturing and preprocessing which are performed on the individual product instances have a huge impact on the quality of the information that is presented to post-processing and analysis.

3.1. Observation system

An observation system (cf. fig. 2) consists of three main layers, (i) the *authoring and analysis layer* where specification of observation and the captured information is worked with, (ii) the *repository layer* which accomplishes the task of configuration and data aggregation, and (iii) the *observation layer* with local product instances. Observation specifications have to transmitted to product instances and observed information has to be captured in a central instance for further analysis. In the optimal case, a knowledge engineer can define an observation specification and the infrastructure provides all services necessary to configure product instances and transport the data back for analysis.

In this and following sections, we will concentrate both on a part of the authoring and analysis layer and the observation layer. The aforementioned specification of observation consisting of (i) hooks, (ii) processing, and (iii) export, can be modeled by a visual language using few

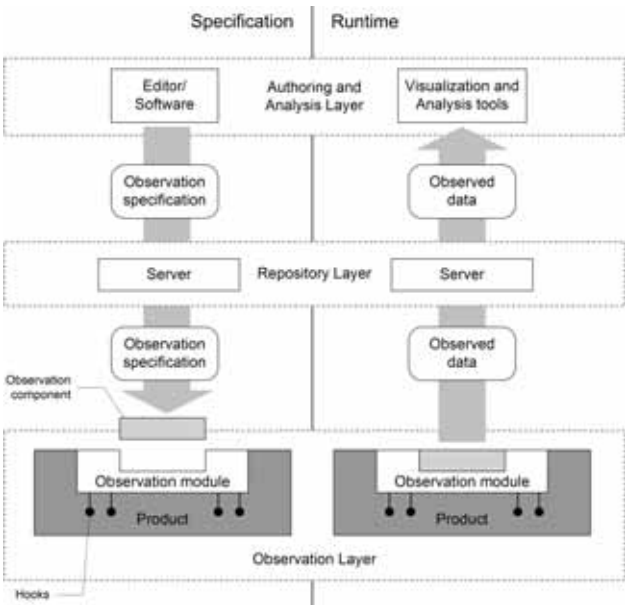


Figure 2. Observation system overview

graphical building blocks (see section 3.2). On the observation layer a runtime structure called *observation component* (OC) is constructed from a visual observation specification by means of model interpretation. An OC is a pluggable part of the observation module that is connected to a specialized runtime environment inside the observation module (cf. fig. 2).

The development effort for a working observation system as a whole can be split up into two main tasks: (i) integration of observation into systems including a middleware capable of information delivery between the observation layer and the repository layer, and (ii) the definition of observation via the visual language. This separation of concerns supports the roles involved in the process: *product developers* handle the first task and *information stakeholders* define what should be observed. Ideally, a third role comes in, the observation developer, who shoulders the burden of observation-specific programming which includes, for instance, the infrastructure, editor customizations and platform-specific adaptations of the observation module.

3.2. Visual language

Observation specification should be performed by experts in the domain of user-related information or other product information stakeholders. Often those people do not have the necessary system engineering and programming skills to instruct a distributed system of product instances. Therefore we propose a visual specification language that hides low-level programming matter and enables domain experts to take advantage of their special knowledge about product information. It is a domain-specific language that focusses on observation only. Likewise, concepts of general purpose programming languages which are inappropriate for the specification can be left out. The essential language elements shall be described in the following.

Hooks are places for information retrieval and they are basically the only information inlets of the observation system. There are two types of hooks, the ones that have to be triggered to yield data, and the ones which trigger themselves and can be seen as manifestations of events in the over-all system. As event generators, hooks are also the sole platform-specific parts of the system and represent an interface between the product's internals and the observation module. The hooks that are not self-triggering can be linked to *timers* which simply realize periodic signals that cause those hooks to fire. This sampling technique is used especially for information like performance measurements or resource load that has a continuous nature.

Hooks generate low-level system data that is mainly not immediately useful for analysis. It is a mass of atomic system events, that has no inherent structure and does not gain any comprehensive results - let alone answering specific questions. Therefore this data has to be preprocessed to become meaningful. The next stage of the observation specification tackles this aspect. Hook data is routed through *processing blocks*. Processing can involve calculations to normalize incoming numbers or the correlation of multiple events to gain derivative complex events. Closely related are *caching blocks* that enable data snapshots and can, for instance, be used as a sliding window over an event stream to compute a floating average.

After finishing those early computations the information shall be exported. This means to transfer it from the product instance to the repository layer that gathers all product usage data in a central data storage. For this purpose the visual language contains an outlet symbol, which can be used to route outgoing information and to label the information according to the semantics it represents.

All aforementioned visual blocks can be linked by *routes* which connect the outlet of a block with inlets of other blocks, thus forming a directed graph (cf. fig. 4 for an example). An observation specification denotes an event-driven system that reacts on the occurence of events and may also trigger hooks by the use of timers. Still, from the user perspective the flow of information in such a description can be seen relatively easily and the language abstracts from concurrency issues as well as from potential data conversion problems. Its visual form allows to concentrate on the matter of specification on the information level.

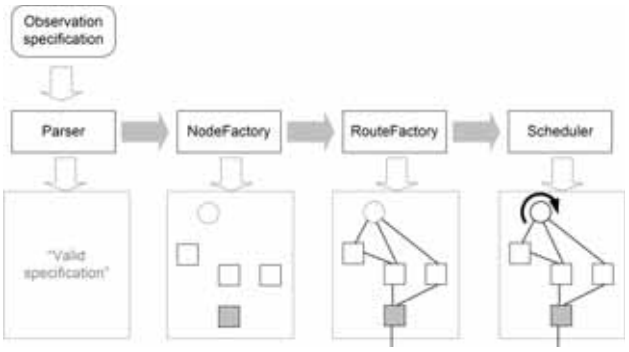


Figure 3. From observation specification to observation component

3.3. Observation module

The visual language's counterpart on the observation layer (cf. figure 2) is a module integrated into the system to be observed. The degree of integration into the host system depends on the required access on information sources. An observation module can be realized in multiple fashions, e.g. as a plugin for existing software, as separate software or even as a dedicated hardware subsystem - as long as the basic requirements (i) access to information sources and (ii) communication capabilities with a repository layer are fulfilled. For less strict requirements, observation modules can accompany a system as long as its services are needed and should be easy to remove after use. Also the impact on system performance and of course privacy as well as security issues have to be considered carefully, but are out of scope here.

As mentioned before, the module provides internal execution capabilities: it receives an observation specification, constructs and runs an OC, and delivers the collected information towards the repository layer. Therefore it consists of several parts that play a role in communication, configuration and the observation itself. The communication subsystem jointly realizes the data transmission infrastructure depicted in figure 2 together with a server on the repository layer. The configuration subsystem essentially contains the parts shown in figure 3 which parse and construct an OC from a specification (further discussed in 4.1). The building blocks of an OC are particularly interesting in the context of the next section and shall be described there.

4. Model interpretation

Model-driven engineering being one of the most influencal achievements in recent system development is also at the core of technical observation development. Especially the agile nature of observation development and iterative characteristics of the process require an automated flow so that changes in observation requirements can be propagated quickly towards actual execution [11]. Furthermore, it is crucial to protect the client machines from potentially harmful virtual machine bytecode or, potentially worse, binary code. Still, the highly dynamic nature of the product evaluation settings demand an special engineering approach: runtime structures are constructed directly from the specification. This technique replaces the transformation and code generation steps of traditional MDE with a single interpretation step. Code generation in principle transforms a model into a textual representation which is processable by, e.g. a compiler. In contrast, model interpretation directly processes the model and generates executable structures in memory. This has the main advantage that the model can be *embedded* into the runtime system. This emphasizes the safety of the system and allows for a change of the observation behavior at runtime, simply by replacing the interpreted model with a newer version.

4.1. Observation specification

For the specification of observation we developed an editor based on the Eclipse platform. That editor allows for an easy composition of an observation specification suitable for domain experts. Also, it offers the possibility to send the finished specification directly to a server on the repository layer which is part of the distribution infrastructure for updating product instances. Figure 4 shows an example of a graphical observation specification. It denotes the timed triggering of a hook requesting information about the CPU performance every ten seconds. This information is averaged (*avg* symbol) by a processing node and exported via an export node.

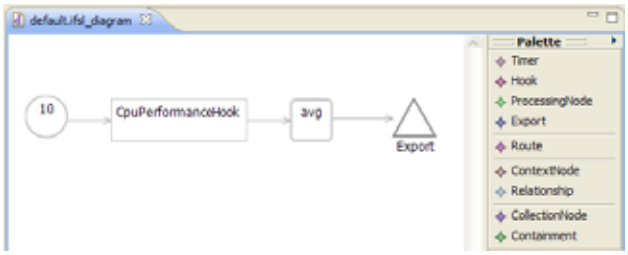


Figure 4. Visual editor screenshot with an example specification

Models expressed in the visual language are serialized in plain XML. This can be parsed by the observation module. Corresponding to the example specification shown in figure 4, its structure also appears in the XML file that is sent to the observation module for execution (cf. figure 5).

```
<?xml version="1.0" encoding="UTF-8"?>
<ifsl:Model xmi:version="2.0">
  <elements type="ifsl:Timer" period="10"
      unit="seconds"/>
  <elements type="ifsl:PlatformHook" name="
      CpuPerformanceHook"/>
  <elements type="ifsl:Route" end1="//
      @elements.0" end2="//@elements.1"/>
  <elements type="ifsl:ProcessingNode" name
      ="avg"/>
  <elements type="ifsl:Route" end1="//
      @elements.1" end2="//@elements.3"/>
  <elements type="ifsl:XMLExportNode" name=
      "Export"/>
  <elements type="ifsl:Route" end1="//
      @elements.3" end2="//@elements.5"/>
</ifsl:Model>
```

Figure 5. XML version of observation specification

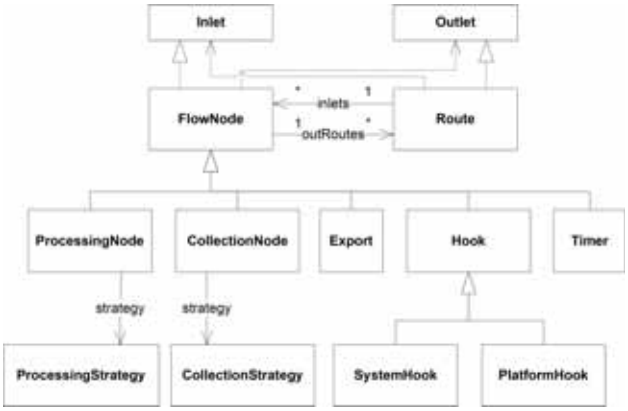


Figure 6. UML class diagram of observation component building blocks

4.2. Observation runtime

The observation module contains a runtime environment that can execute an OC as specified by the visual model. Figure 3 shows the configuration process. An observation specification is parsed and checked for validity. After that, a set of building blocks is constructed dynamically using a *NodeFactory*. The routing unit takes this set of blocks as input and creates routes according to the specification. Implicitly the export nodes of the network are connected to the respective communication facility. Finally, the scheduler subsystem starts all timers, the only active parts in an otherwise reactive event-based architecture.

To construct such a network, the event-driven executable system makes use of the base class *FlowNode* which realizes the basic routing functionality together with the *Route* class. As the UML diagram (see figure 6) shows, dynamic linking of nodes is accomplished by using the *FlowNode-Route-Inlet-Outlet* pattern: *FlowNodes* offer inlet functionality by means of a provided interface and a *Route* can connect to those nodes with a usage relationship with the interface *Inlet*. For the interface *Outlet* the reversed relationships hold. Objects are linked together by means of the *inlets* and *outRoutes* associations.

The UML class diagram in figure 6 depicts also the classes that represent hooks, processing nodes, collection nodes, exports and timers. Obviously there is a 1:1 relationship between elements of the visual language and the instanciated objects that are linked together by means of the inherent routing functionality of all objects derived from the base class. What the picture also shows is the application of the strategy pattern [9]. It helps to realize different kinds of behaviour of *ProcessingNodes* and *CollectionNodes*. Depending on the type of processing or collection specified, different strategies can be chosen. This especially reduces the effort for observation development as only necessary computations have to be implemented in the observation module.

Yet, the implementation of hooks proved to be the most demanding task of observation development as it means to interface the product at various levels, a task that strongly depends on documentation and openness of the platform.

There are basically two types of hooks: Platform hooks which access the host system and are characterized by mainly platform-specific behaviour, thus the name, and system hooks that access the observation module. While the use of platform hooks for information collection is straightforward, system hooks capture events concerning the observation itself. In the future, this can be used to adapt the observation to context changes or to establish semantic links between observed items on multiple levels.

5. Case study

Together with a large Dutch electronics company we carried out a case-study to test the observation of a consumer electronics prototype. This showed the applicability of the approach in a world-wide observation scenario that connected 20 machines spread over 8 countries to a central server which collected about 800.000 data items. The product instances were pre-configured before roll-out. As expected, changes in the observation requirements of information stakeholders demanded for remote changes of the observation specification which were performed successfully several times. The machines continued to capture data according to a new observation specification.

The data collected during the case-study supported mostly the assumptions of the development team about product usage, but also gave new insight on country-specific usage problems and customer expectations. Regarding the successful application of the proposed technology and new insight into product usage it has been decided to continue with successive experiments on a later version of the observed product prototype.

6. Conclusion & Future work

Regarding the fact that the majority of products are currently not designed for observation, we are working in the direction of a design method for self-observing systems. To provide an intermediate solution for observation integration we chose model interpretation for maximum flexibility and agility. Our approach is to specify visually and to execute the finished specification directly on the product. This emphasizes the separation of concerns between domain experts who are interested in the collection of usage information and developers who are concerned with the system engineering.

We developed an experimental framework for specification and implementation of observation functionality. A new visual specification language has been introduced to support domain experts specifying observation behavior. It proved to greatly simplify the task of product usage data collection. The language is generic enough to be reused in different observation contexts. Only minor changes have to be made to the observation specification runtime environment in case a new product software implementation platform has to be entered.

The case-study showed that as soon as observation modules are in place and the specification supports basic measurements of users' interactions with a product the need for better semantic linking between observed data arises. So far, a lot of effort still has to be spent on the post-processing of captured data. The next step is an annotation of events with semantic information that tells e.g. about the origin, conditions and context of such an event. Also this is done in a structured way, such that the information can be easily exploited during post-processing using automatic analysis tools. Another future direction is the collection and incorporation of subjective user feedback data into the set of objective product data. In addition, subjective data can help to understand the *why* in user-product interaction. It will be possible to bind subjective feedback measures to the occurence of certain events which enables a dynamic insight into the usage process together with background information coming directly from the user at the same time.

Acknowledgments

This work is being carried out as part of the "Managing Soft-Reliability in Strongly Innovative Product Creation Processes" project, sponsored by the Dutch Ministry of Economic Affairs under the IOP-IPCR program.

References

[1] S. Bly, B. Schilit, D. W. McDonald, B. Rosario, and Y. Saint-Hilaire. Broken expectations in the digital home. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 568–573, New York, NY, USA, 2006. ACM Press.

[2] R. Cooper and E. Kleinschmidt. New products: What separates winners from losers? *Journal of Product Innovation Management*, 4, September 1987.

[3] E. den Ouden, L. Yuan, P. J. M. Sonnemans, and A. C. Brombacher. Quality and reliability problems from a consumer's perspective: an increasing problem overlooked by businesses? *Quality and Reliability Engineering International*, 22(7):821–838, 2006.

[4] M. Diep. Profiling deployed software: Assessing strategies and testing opportunities. *IEEE Trans. Softw. Eng.*, 31(4):312–327, 2005.

[5] J. Estublier and G. Vega. Reuse and variability in large software applications. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 316–325, New York, NY, USA, 2005. ACM.

[6] E. Evans. *Domain Driven Design*. Addison-Wesley, 2004.

[7] D. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[8] M. Funk, P. van der Putten, and H. Corporaal. Specification for user modeling with self-observing systems. In *Proceedings of the First International Conference on Advances in Computer-Human Interaction*, 2008.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 1994.

[10] K. Kabitzsch and V. Vasyutynskyy. Architecture and data model for monitoring of distributed automation systems. In *1st IFAC Symposium on Telematics Applications In Automation and Robotics*, Helsinki, 2004.

[11] J. Karsai, G.; Sztipanovits. A model-based approach to self-adaptive software. *Intelligent Systems and Their Applications, IEEE [see also IEEE Intelligent Systems]*, 14(3):46–53, May/Jun 1999.

[12] A. Kobsa. Generic user modeling systems. *User Modeling and User-Adapted Interaction*, 11(1):49–63, Mar. 2001.

[13] J. Kort and H. de Poot. Usage analysis: combining logging and qualitative methods. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 2121–2122, New York, NY, USA, 2005. ACM Press.

[14] B. Selic. The pragmatics of model-driven development. *Software, IEEE*, 20(5):19–25, 2003.

Network Intrusion Detection Based on Bayesian Networks

Alma Cemerlic, Li Yang, Joseph M. Kizza
Department of Computer Science and Engineering
University of Tennessee at Chattanooga
Chattanooga, TN 37403
Alma-Cemerlic@utc.edu, Li-Yang@utc.edu, Joseph-Kizza@utc.edu

Abstract

Intrusion detection has drawn much attention in the past two decades. Signature analysis and statistical anomaly detection are two typical methods to identify network security breaches. Signature analysis requires access to a large database of known intrusion signatures and a way to match current behavior against the signatures to detect intrusions in progress. The limitation of this approach lies in its dependence on frequent updates of the signature database and its inability to generalize and detect novel intrusions. Anomaly detection methods can detect attacks based on statistical probability, which allows for generalization and helps in detection of novel attacks. However, statistical anomaly detection is not based on an adaptive intelligent model and cannot learn from normal and malicious traffic patterns. We propose an adaptive network intrusion detection using a Bayesian network, trained with a mixed dataset containing real-world and DARPA dataset traffic. Our Intrusion Detection System (IDS) model is designed to detect novel attacks. We use features of network connections to parameterize the system. The DARPA dataset and real-world traffic are used to measure the feasibility and effectiveness of our system. The network connections that are confirmed to be novel intrusions are added to the training dataset to re-train our IDS, thus enhancing our system's ability to detect future intrusions.

Keywords: Intrusion detection, Bayesian network

1. INTRODUCTION

Today, a large amount of sensitive information is processed through computer networks, thus it is increasingly important to make information systems, especially those used for critical functions in the military and commercial sectors, resistant and tolerant to network intrusions. An intrusion can be defined [9] as an attempt to gain unauthorized access to network resources. As the number of newly discovered vulnerabilities per year increases and as hacker tools become more advanced and automated, intrusion prevention techniques alone are not sufficient. Today, Intrusion Detection Systems (IDSs) are necessary for effective computer system protection. An intrusion can be detected using either signature-based detection or anomaly-based detection. Signature-based analysis [8] as an intrusion detection technique requires a database of signatures of known intrusions in order to be able to detect attacks. The key advantage of signature detection techniques is in their high degree of accuracy in detecting known attacks and their variations. The main drawbacks are the need to frequently update the database of intrusion signatures and the inability to generalize and detect novel intrusions. In addition to these drawbacks, even if a new attack is discovered and its signature determined, there is often a substantial latency in the update of the signature databases for IDSs across networks.

Anomaly detection techniques based on statistics, such as IDES [7], send an alarm when they detect an event that deviates from the behavior defined as *normal*. The observed network traffic is compared to profiles of normal network use. Statistical anomaly detection has no intelligent learning model which may lead to a high rate of false alarms. This happens primarily because previously unseen (yet legitimate) system behaviors may also be recognized as anomalies and hence flagged as potential intrusions. All these limitations have led to an increasing interest in intrusion detection systems based on data mining.

Several researchers have been interested in developing IDSs using a generalization learning model. Axelsson et al. [12] employ Bayesian inference steps with transition models between inferences to assess whether a particular burst of traffic contains an attack. Kruegel et al. [10] proposed a model which simulates an intelligent attacker using Bayesian techniques to create a plan of goal-directed actions. This study also proposes an event classification scheme based on Bayesian networks. The advantage of Bayesian networks is in that they improve the aggregation of different model outputs and allow one to seamlessly incorporate additional information into an already existing model. Johansen et al. [11] believe that a Bayesian system provides a solid mathematics foundation to simplify a seemingly difficult and monstrous problem that today's IDS implementations fail to solve. They added that Bayesian network IDS should differentiate between attacks and the normal network activity by comparing metrics of each network traffic sample.

We propose to develop an adaptive network intrusion detection system using a Bayesian network (BN), trained with a mixed dataset containing real-world and DARPA dataset traffic, aiming to detect novel intrusions with low number of false alarms. Our proposed IDS model is able to parse real-world traffic and identify network attacks including novel attacks that the system has not previously encountered. A BN is used to build an automatic intrusion detection model and signal an intrusion when a suspicious activity is noticed.

2. FRAMEWORK OF AN ADAPTIVE IDS

The architecture of our proposed intrusion detection system consists of six modules (Fig. 1). The *Data gathering (sensors) and parsing* module is responsible for collecting data from the monitored network and parsing them into connections. A connection is equivalent to a session between two hosts on a network, and it is composed of all the observed packets that the hosts exchanged. The *Bayesian Network Inference* module is the analysis engine of the IDS responsible for processing the data collected from the sensors. The *Knowledge base* contains an intelligent model (Bayesian network) which learns from observed traffic and has the ability to predict whether a network connection is an attack. The *System configuration* provides information about the current state of the IDS. The *Response* component initiates actions when an intrusion is detected. The responses can either be automated (active) or involve human interaction (inactive). The *Bayesian Network Learning* module is used to build up knowledge from the offline training dataset.

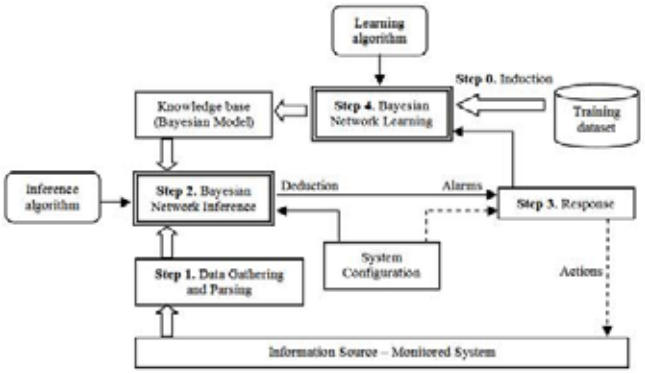


Fig. 1 Bayesian Network-based IDS architecture

Overall, our proposed framework consists of a training component and a detection component. We use a training dataset to parameterize the IDS. The DARPA dataset and real-world traffic are used to measure the feasibility and effectiveness of our system. Given the training dataset, the training component estimates the parameters of the Bayesian model in *Step 0* in Fig. 1. The network traffic is gathered and parsed into application layer network connections in *Step 1*. The Bayesian model then considers both the network connections and the system configuration to infer the probability that the network is under attack. The Bayesian model is built based on a learning algorithm using training data as the knowledge base, shown in *Step 2*. A network connection recognized as an intrusion will trigger an IDS response. A never-before-seen network connection is marked as suspicious if it is classified as intrusive by our IDS system. If the suspicious connection is confirmed to be an intrusion by a network administrator, it is added to the training dataset to re-train our adaptive IDS, as shown in *Step 4*. This process enhances our system's ability to detect future intrusions.

3 BAYESIAN NETWORK

A Bayesian network is a graphic representation of the joint probability distribution function over a set of variables. The network structure is represented as a Directed Acyclic Graph (DAG) in which each node corresponds to a random variable and each edge indicates a dependent relationship between connected variables. Each variable (node) in a BN is associated with a Conditional Probability Table (CPT), which enumerates the conditional probabilities for this variable given all the combinations of its parents' values [2]. Therefore, for a BN, the DAG captures causal relationships among random variables, and CPTs quantify these relationships. Since individual events in an attack can be represented as nodes and the causal relations between events can be modeled as edges in Bayesian networks, we use a BN as our inference model. A BN model is capable of learning causal relationships from an existing dataset and predicting the consequences of an intervention in the problem domain. A BN is an ideal model for combining prior knowledge with new data and inferring posterior knowledge.

In order to learn the structure and test our proposed BN with datasets, we use *Netica* and *Genie*, the tools for modeling BNs.

3.1 Learning Algorithm

In our model, we use the K2 *learning algorithm.* The algorithm defines a set of variables of interest to build a directed acyclic graph (DAG) based on the calculation of a local score [6]. K2 is initialized with a single node, and it continues to incrementally add connections with other nodes as long as they increase the whole probability of the network structure. We use the following network connection features ordered according to the relevance analysis in [4]: *protocol_type*, *sevice*, *num_wrong_fragments*, *land*, *logged_in*, *num_failed_login*, *root_shell*, *is_guest_login*, and *type*.

3.2 Inference Algorithm

For the inference in our model, we use the Junction Tree Algorithm [5]. The idea behind this procedure is to construct a data structure called a junction tree which can be used to calculate any query through message passing on the tree. To build a junction tree, we first choose an ordering of the nodes and use node elimination to obtain a set of elimination cliques. A complete cluster graph is then built over the maximal elimination cliques. Each edge {B, C} is weighted by $|B \cap C|$ to compute a maximum-weight spanning tree. This spanning tree is a junction tree.

4. INTRUSION DETECTION TESTING DATASET

We use two different datasets to test our proposed IDS model, namely the DARPA dataset and the real network traffic collected in our security lab.

4.1 DARPA Dataset

The DARPA intrusion detection evaluation dataset [1] from MIT Lincoln Lab is used to train and test our IDS. The dataset was collected from a simulation of a fictitious military network over the period of seven weeks.

Before feeding the data to the Bayesian network, for either learning or testing, raw network traffic has to be pre-processed and summarized into connections or high-level events. Each connection is described with a set of features. The DARPA KDD 99 dataset summarized DARPA 98 Lincoln Lab network traffic into connections with 41-features per connection. We define a connection as a sequence of TCP packets starting and ending at some well defined points in time, between which data flows from a source IP address to a target IP address under some well defined protocol. In our model, we use 9 of the 41 features, namely *protocol_type*, *service*, *num_of_wrong_fragments, num_of_failed_logins*, *land ,login_success, is_guest_login, root_shell_obtained,* and *type* (intrusion or normal connection).

Netwotk intrusions are classified into four categories [1]: *user-to-root (u2r)*, *remote-to-local (r2l)*, *denial-of-service (DoS)*, and *probe*. The u2r attack occurs when attackers who have local access to the victim machines try to gain superuser privileges. The r2l attack happens when attackers who have no account on the victim machine try to gain access. The DoS attack occurs when attackers try to prevent legitimate users from using a service available on the network. The goal of a probe attack is to gain information about the target host.

In the case of signature-based IDSs, the recency of the data in the signature database is crucial. In our case, the recency of the dataset is not significant because our model is an anomaly detector that needs no specific knowledge about attacks. Thus, the DARPA dataset is still viable for testing our model [13].

We used the labeled training dataset to *train* our Bayesian model, and the testing dataset to *test* for the correct discovery of intrusions. The Bayesian network used in our IDS model is shown in Fig. 2.

4.2 Customized Dataset with Novel Intrusions

After learning the BN model and testing it using the DARPA dataset, we created a custom dataset to measure the capability of BN model in detection of never-before-seen attacks. The custom dataset contains both attacks and normal traffic. The attacks are collected through repeating the vulnerability exploits available in the Metasploit 3 framework [3]. The traffic containing these attacks is recorded in the form of tcpdump files. We chose exploits in a way that they represent all four general categories of attacks: DoS, r2l, u2r, and probe [1].

A type of attacks known as buffer overflow (BoF) attacks can be used to gain the root access on the victim system. Depending on the targeted platform, a buffer overflow attack can be executed as u2r or r2l. For instance, the *Microsoft Plug and Play Service Overflow*, which exploits the plug and play service used by the operating system to detect new hardware, is an example of a buffer overflow attack that on certain platforms requires local access to be successfully completed, while on others can be executed remotely. Additionally, under certain circumstances, buffer overflow attacks can result in a DoS attack. For example, *NetpwPathCanonicalize Overflow* in Microsoft Server Service exploits a stack overflow in the *NetApi32 CanonicalizePathName* function using a Remote Procedure Call (RPC) call in the Server Service. On certain Windows platforms, even if unsuccessful, this attack can cause termination of all SMB-related services or a system reboot, and thus is classified as a DoS attack. As an addition to the set of probe attacks, we collected a footprint of a UDP service sweeper, a tool designed to detect common UDP services available on the target host.

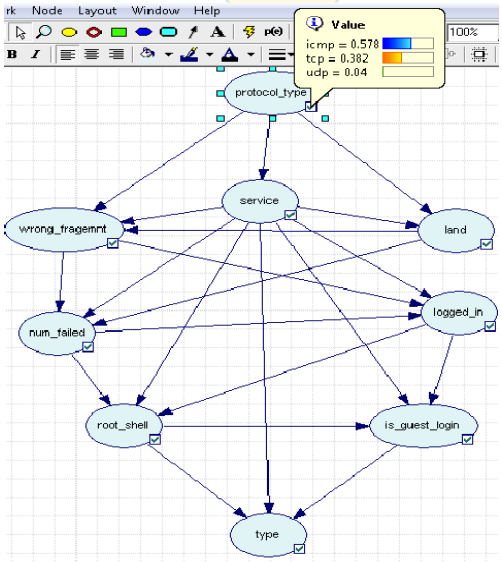


Fig. 2 Learned Bayesian Network for IDS

5. EXPERIMENT SETUP

Our experiment consists of two phases. In the first phase, we use the DARPA training and testing datasets to train and test our Bayesian model respectively. In the second phase, we capture the real network traffic to further test the system.

The traffic features relevant to our IDS are associated with each network connection rather than with each individual packet. This results in a faster Bayesian network training and a faster classification of incoming connections as either normal or intrusion. We monitor nine features for each connection. They are: *protocol_type*, *sevice*, *num_wrong_fragments*, *land*, *logged_in*, *num_failed_login*, *root_shell*, *is_guest_login*, and *type*. The last listed feature is used to label a specific connection as either normal or an attack for the purpose of training and testing.

The initial learning and testing data sets are composed of labeled DARPA 98 dataset records.

The real-world network traffic serves to test the system's ability to recognize never-before-seen attacks. The traffic collected in the tcpdump format is preprocessed by a custom parser which first groups the packets into connections, then extracts connection-specific features we use in our model. Eight of the features are extracted from packet headers and payload, while the nineth feature, *type*, is added by hand.

Once the structure is learned and our Bayesian network is trained, the network is able to examine any input given in the correct format and label each connection as either normal or an intrusion. Those connections that do not fall in either class are labeled as intrusions, since they may be novel attacks.They are included in the learning dataset and used to retrain and improve the network. However, prior to adding the potential novel intrusions to the learning dataset, their classification needs to be inspected for correctness before they are able to affect the BN structure.

6. EXPERIMENT EVALUATION

In order to have a good prediction performance, an IDS should be able to correctly differentiate between intrusions and legitimate actions in a system environment. Typical features for evaluating predictive performance of IDSs include *true positive* (TP) rate (detection rate) and *false positive* (FP) rate shown in Table 1. True positive rate is the ratio of the number of correctly detected attacks and the total number of attacks, and false positive rate is the ratio of the number of normal connections that are incorrectly classified as attacks and the total number of normal connections.

The performance analysis of our IDS given in Table 1 is reported on the 50% cutoff line, which means the Bayesian network classified an event as an intrusion only if its belief was higher than 50%. If we choose to lower this boundary, the percent of TP will rise, but also will the percent of FP.

Our experimental results for the DARPA datasets are as follows: True negative rate correctly recognizing the normal connection is 93.89%. True positive rate correctly determining intrusions is 97.88%.The error rate is 2.881%, which means that in 2.881% of cases the network predicted a wrong value, where the predicted value is the one that had a higher belief value.

	Predicted Normal	***Predicted Intrusions***
Actual Normal Connection	**T**rue **N**egative 93.89%	**F**alse **P**ositive 6.11%
Actual Intrusions (Attacks)	**F**alse **N**egative 1.45%	**T**rue **P**ositive 97.88%

Table 1. Evaluation of Intrusion Detection

When trained only by the DARPA training dataset, our IDS model indicated the presence of new intrusions in real-world testing dataset by conflicting evidences. Only the CAN-2003-0003 exploit and the UDP sweep were correctly detected. The reason is that the DARPA training dataset is much simpler than real-world traffic. The model trained by the DARPA dataset has limited capability to detect real-world normal traffic and attacks. Our solution is to mix the real-world training dataset with the DARPA dataset to train our IDS model again. After re-training, the model is able to correctly recognize the malicious connections it earlier was not able to.

7. CONCLUSION AND FUTURE WORKS

We developed an adaptive anomaly-based IDS to detect unknown attacks. This IDS has been tested with both the DARPA dataset and a real network traffic containing novel attacks. The detection rate was increased after the IDS model was retrained by a dataset that included the correctly labeled real-world traffic.

Since the optimal node ordering with respect to the topology of Bayesian network is NP-hard, and the Bayesian network trained in the standard way does not perform to a satisfactory level, we plan to locally optimize the Bayesian network to improve the effectiveness of our IDS system. We will also add events from the system architecture level (such as CPU utilization) to the application level connection features that we currently use.

Acknowledgements: supported in part by Tennessee Higher Education Commission's Center of Excellence in Applied Computational Science and Engineering under R04-1330-023.

8. REFERENCES

[1] DARPA. Knowledge Discovery in Databases, 1999. DARPA archive. http://www.kdd.ics.uci.edu/databases/kddcup99/task.htm

[2] F. Jesen. Bayesian Networks and Decision Graphs. Springer, New York, USA, 2001.

[3] The Metasploit framework: http://www.metasploit.com

[4] H. G. Kayacik, A. N. Zincir-Heywood, M. I. Heywood. Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99. *Proceeding of third annual conference on privacy, security and trust (PST)*, New Brunswick, Canada. Oct. 2005.

[5] F. Jemili, M. Zaghdoud, M. Ben Ahmed. A Framework for an Adaptive Intrusion Detection System using Bayesian Network. ISI IEEE, 2007.

[6] D. Barber. Machine Learning: A Probabilistic Approach, pg.107, 2007.

[7] T. Lunt, A.Tamaru, F. Gilhan, R.Jaganathan, P. Neumann, H. Javitz, A. Valdes, and T. Garvey. A real-time intrustion detection expert system (IDES). Technical report, Computer Science Laboratory, SRI International, Menlo, Park, California, Feb. 1992.

[8] K. Ilgun, R. A. Kemmerer, P.A. Porras. State transition analysis: A rule-based intrusion detection approach. IEEE Transaction on Software Engineering, 21(3): pg. 181-199, Mar. 1995.

[9] R. Heady, G. Luger, A. Maccabe, M. Servilla. The architecture of a network level intrusion detection system. Technical report, Computer Science Department, University of New Mexico, Aug. 1990.

[10] C. Kruegel, D. Mutz, W. Robertson, F. Valeur. Bayesian event classification for intrusion detection. *Proceedings of the 19th Annual Computer Security Applications Conference*, Las Vegas, NE, Dec. 2003.

[11] K. Johansen, S. Lee. Network Security: Bayesian Network Intrusion Detection (BINDS) May, 2003.

[12] S. Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transaction of Information System Security* 3, 3 (Aug. 2000), pg. 186-205.

[13] M. Mahoney, P.K. Chan. An analysis of the 1999 DARPA/Lincoln laboratory evaluation data for network anomaly detection. Recent advances in intrusion detection: 6th international symposium. RAID, pg. 220-237.

Supremum of Agent Number Needed in Analyzing Security Protocols Based on Horn Logic

Feng Liu
School of Computer Science
National University
of Defence Technology
Email: baichengmeizi@163.com

Zhoujun Li
School of Computer
BeiHang University

Ti Zhou
and Mengjun Li
School of Computer Science
National University
of Defence Technology

Abstract

To study the supremum of the number of agents needed in analyzing security protocols for properties such as secrecy and authentication, an extended trace model for security protocols and properties based on Horn logic program is introduced. Strategy vectors are added to Horn logic programs for protocols, which ensure that the intruder can have nondeterministic choices, and help to correctly and completely describe the reception of messages and consequently cover more attacks. The supremum of the number of honest agents needed to express an attack on a protocol $\mathcal{P}$ for secrecy property and authentication property is given as $N_{\mathcal{P}}$(the number of roles except servers in $\mathcal{P}$), which is fixed for each protocol. The supremum of the number of dishonest agents needed to express an attack on a protocol $\mathcal{P}$ is proved to be dependent on whether the security property being analyzed is considered as strong or weak.

Keywords
security protocol; security property; Horn logic; role; agent

I. Introduction

Security protocols are designed for distributing secret data items, authenticating users, and even accomplishing electronic transactions[3]. Security protocols, however, are not always competent enough. Many protocols have been shown having flaws[3][10]. Utilizing these flaws, intruders can get secret data items shared by honest agents, or impersonate some honest agents to cheat others. Accordingly, several properties of security protocols have been defined: secrecy, authentication, fairness, accountability, etc[3].

The analysis and verification of security protocols has gained many researchers' attention during past years. There have been lots of work concerning protocol flaws. Many techniques are adopted, including both informal and formal ones.

Security protocols can be described informally as transfer of messages between roles. A protocol contains fixed roles and steps. The number of roles except servers of a protocol $\mathcal{P}$ is denoted as $N_{\mathcal{P}}$. Each step involves a message transfer between two roles. For example, SPLICE/AS authentication protocol is described informally in Fig 1, here $N_{\mathcal{P}} = 2$ (AS is considered as an authentication server).

Fig. 1. The informal description of SPLICE/AS authentication protocol

$(1) C \rightarrow AS \ : \ C, S, N_1$
$(2) AS \rightarrow C \ : \ AS, E(K_{AS}^{-1} : AS, C, N_1, K_S)$
$(3)\ C \rightarrow S \ : \ C, S, E(K_C^{-1} : C, T, L, E(K_S : N_2))$
$(4) S \rightarrow AS \ : \ S, C, N_3$
$(5) AS \rightarrow S \ : \ AS, E(K_{AS}^{-1} : AS, S, N_3, K_C)$
$(6)\ S \rightarrow C \ : \ S, C, E(K_C : S, N_2 + 1)$

In analyzing security protocols, there are several important parameters that determine the complexity of the models of protocols:

-The number of runs (i.e., sessions).

-The number of agents taking parts in the environment.

-The nonces and other fresh values that are produced by agents at run-time.

-The length and depth of messages.

At run-time, roles are instantiated by different agents. A protocol can be run for many times(sessions) by different agents randomly. Each session is identified with a session number. Different sessions may be concurrent or even interleaving. Nonces can be considered as parameterized by agent IDs and session numbers.

Until now, the supremum of the number of sessions is still open, and Blanchet has adopted some approximation in deduction to deal with infinite sessions[5][6].

There are infinite agents in the run-time environment of protocols, including both honest agents and dishonest agents. Each agent has his own ID and keys. Furthermore, dishonest agents are controlled by an intruder. So here comes one decision-making problem: how many dishonest agents is required for analyzing a protocol? On the other hand, how many honest agents should be required? This leads to another decision-making problem. The two decision-making problems are what we want to work on in this study.

There have been some work on the supremum of the number of agents in analyzing protocols. Even has given a definite supremum of the number of dishonest agents to find

Supported by the National Natural Science Foundation of China under Grant No. 60473057, 90604007.

attacks in Ping-Pong protocols[8][9]. Comon has studied the supremum of agents needed in analyzing security properties on a restricted model for protocols and intruders, and proven that any attack can be converted into an attack involving some *k* agents[4], whose supremum is not given clear.

We have improved the trace model in [4] to allow non-deterministic choices for the intruder, and the reception of messages is expressed more completely by strategy vectors for the intruder. Without this improvement, there would be attacks not able to be covered in the original model. The improvement works well with the original model, since strategy vectors can be conveniently applied into preconditions of Horn clauses. By regulating the number of roles of protocols and agent variables in property clauses and modifying the projection, we determine that the supremum for the number of honest agents required in analyzing a protocol $\mathcal{P}$ is $N_{\mathcal{P}}$, which is fixed for each protocol. The supremum for the number of dishonest agents is proven to be 1 or open depending on whether a dishonest agent can participate in a protocol session as more than 2 roles. We strictly and systematically concentrated on semantical relationship between protocol programs and property clauses. Our proof is based on a projection on not only Herbrand universes but also Herbrand bases which reduces the number of agents while preserving the semantics of attacks. The definition and proof of semantical preservation of projections make it easy to apply the result to other security properties besides secrecy and authentication by only adjusting the property clauses.

The paper is organized as follows: Section II describes the model for protocols and properties. Section III describes the semantical relationship between protocols and properties in this model. Section IV describes the projection on Herbrand base and the semantics of attacks under this projection. Section V is the conclusion and plan of future work.

II. Representation of Security Protocols and Properties

In this section, we introduce a model for security protocols. We adopt Comon's trace model based on logic program[4], but some improvements and rectifications have been done in order to represent run-time protocols more accurately and cover protocol properties more extensively.

In this model, a security protocol is formalized as a Horn logic program, i.e., a set of Horn clauses. Properties of the protocol are also formalized as Horn clauses. The reason for basing the model on logic program is that it's convenient to deal with the semantics of the protocol. Logic programs have Herbrand models, on which we can make projection and induction on agents.

This model is a trace model. A sequence of actions performed by the agents, maybe in different sessions, compose a trace.

A. Terms: Message, Event and Trace

A protocol contains several messages transferred between agents. Sending or receiving of a message in a session is considered as an event. A trace is a sequence of events accompanied by a session identifier.

Message, event and trace are represented by terms. The basic sorts are: **Num**, **Agent**, **Message**, **Event**, **Trace**.

- **Num** is an infinite set of numbers . 0∈**Num** is a constant, *S*() is a function on **Num**. If *n*∈**Num**, then *S*(*n*)∈**Num**.
- **Agent**=**Ha**∪**Da**∪**Server**. **Ha** is the set of honest agents, **Da** is the set of dishonest agents, **Server** is the set of constant servers. h∈**Ha**, d∈**Da** are constants. $S_h()$, $S_d()$ are function symbols on **Ha**, **Da**. If x ∈**Ha**, y ∈**Da**, then $S_h(x)$ ∈**Ha**, $S_d(y)$ ∈**Da**. $S_h()$, $S_d()$ are useful in Herbrand interpretations for providing infinite agents in Herbrand universe.
- Most protocols involve nonces (fresh values and time stamps). Nonces in different sessions must be different, hence they are parameterized by agents and session numbers, e.g., $m(a_1, .., a_{N_{\mathcal{P}}}, s)$, s is the session number, $a_1, .., a_{N_{\mathcal{P}}}$ are the roles, $N_{\mathcal{P}}$ is the number of roles involved in the protocol $\mathcal{P}$. The roles are instantiated with agents in a session of $\mathcal{P}$.
- There are several basic and cryptographic function symbols on **Message**, which also construct keys and composite messages: $\{< \_, \_ >, pub(\_), prv(\_) , shr(\_)\}$. The keys are used for encryption and decryption. The term $\{x\}_{pub(y)}$ means that *x* is encrypted by the public key of the agent *y*.
- **Agent**⊂**Message**⊂**Event**, **Num**⊂**Message**⊂**Event**. **Event** contains **Message**, and there is a function symbol $ST(\_, \_, \_, \_)$ to construct events. For example, let a ∈**Agent**, i ∈**Num**, j ∈**Num**, m ∈**Message**, then $ST(a, i, j, m)$∈ **Event**. This term denotes that the agent a is at the jth step, taking the role *i*, and having the message *m* in his memory.
- A trace is a sequence of events accompanied by a session identifier. ⊥ is the empty trace. If t∈**Trace**, e is an event, s is a session number, then $[e, s]$–t ∈ **Trace**. For example, $[e5, s2]$–$[e3, s1]$–$[e2, s]$–$[e1, s]$–⊥ ∈ **Trace**.

B. Predicate & Clause

In this section, we consider clauses which compose a protocol. Horn clause involves literals, which are predicates or their negative forms.

1) Predicate.: For every sort, there is an unary predicate to claim that a variable or a constant belongs to this sort. For example, *Ha*(x) and *T*(t) denote that x is a variable of sort **Ha** and t is a **Trace** variable.

The main predicates are as follows:

- ≠(x,y) denotes the number or agent x is different from y.
- =(x, y) denotes the number x is equal to y.
- *In*$([e, s], t)$ denotes $[e, s]$ occurs in the trace t.
- *Fresh*(t, s) denotes the session number s is fresh in the trace t, that is, s is different with any s' in t.
- *Notplayed*(a, i, s, t) denotes the agent a hasn't perform its ith action in the session s in the trace t.

- $I(x,t)$ denotes the intruder can learn information x from the trace t.
- *Accept*(a,m,s,t) denotes the agent a has checked the message m and decides to accept it in the session s in the trace t.

We extend the domain of the predicate $\neq(x,y)$ so that it can be parameterized by agent variables, and the resulting semantics is that $S_h^i(h)$ differs from $S_h^j(h)$ if $\neq(i,j)$ and $S_d^i(d)$ differs from $S_d^j(d)$ if $\neq(i,j)$. This extension is useful in Horn clauses to keep the agents pairwise different in a session.

2) Protocol Independent Clause.: The clauses for a protocol are divided into two subsets: the protocol independent clauses and the protocol dependent clauses.

Note that in most of the following clauses, the universal quantifiers are omitted for space limitation, that is, there are indeed universal quantifiers at the head of the clauses for each restricted variables.

Protocol independent clauses describe the environment of protocols and the ability of the intruder. These clauses are universal for all protocols.

The clauses that describe the environment involve the assertion of the constants' sort and the deductive relation of some predicates:

$\rightarrow\ Ha(S_h^i(h)), \quad i \geq 0;$
$\rightarrow\ Da(S_d^i(d)), \quad i \geq 0;$
$Num(x)\ \rightarrow \neq(S(x),0);$
$\neq(i,j)\ \rightarrow \neq(S^i(x),S^j(x)), \quad i,j \geq 0;$
$\rightarrow\ In([e,s],\ [e,s]\text{–}t);$
$In([e,s],\ t)\ \rightarrow\ In([e,s],\ [e',s']\text{–}t);$
$\rightarrow\ Fresh(\perp,s);$
$Fresh(t,s),\ \neq(s,s')\ \rightarrow\ Fresh([e,s']\text{–}t,s);$
$\rightarrow\ Notplayed(a,i,s,\perp);$
$Notplayed(a,i,s,\perp),\ \neq(s,s') \rightarrow Notplayed(a,i,s,[e,s']-t)\ ;$
$Notplayed(a,i,s,\perp),\neq(i,i') \rightarrow Notplayed(a,i,s,[ST(a,j,i',m),s]\text{–}t);$

The intruder's ability and cryptographic assumption in this model comply with Dolev-Yao model[2]. The ability of the intruder includes intercepting, detaining, memorizing, analyzing and synthesizing messages. The clauses that describe the ability of the intruder involve the following initial knowledge and computation:

$Agent(x)\ \rightarrow\ I(x,t);$

$Agent(x)\ \rightarrow\ I(pub(x),t);$
$Da(x)\ \rightarrow\ I(prv(x),t);$

$Da(x)\ \rightarrow\ I(shr(x),t);$
$I(<x,y>,t)\ \rightarrow\ I(x,t);$

$I(<x,y>,t)\ \rightarrow\ I(y,t);$
$I(x,t),I(y,t)\ \rightarrow\ I(<x,y>,t);$
$I(\{x\}_{pub(y)},t),\ I(prv(y),t)\ \rightarrow\ I(x,t);$
$I(\{x\}_{prv(y)},t),\ I(pub(y),t)\ \rightarrow\ I(x,t);$
$I(\{x\}_y,t),\ I(y,t)\ \rightarrow\ I(x,t);$
$\rightarrow\ I(x,[x,s]\text{–}t);$

$I(x,t)\ \rightarrow\ I(x,[y,s]\text{–}t);$

For the ability of the intruder, one of the most important is the discrimination between intercepting and detaining, which is not successfully considered in [4]. Intercepting and detaining have the same precondition but different effects. Intercepting means the intruder just learns information from the message or rewrites it, and then let it go on towards the destined receiver. Detaining means that the intruder not only learns information from the message, but also detains it from the destined receiver.

In [3], there are many examples of attacks which rely on detaining, otherwise they couldn't become attacks since the original messages will eventually come to the destined receiver. For example, the Otway-Rees protocol(so does SPLICE/AS authentication protocol) has type flaw attacks which require the intruder to detain some messages[3]. One attack can be partly shown as:

(1) $A \rightarrow Z(B)\ :\ M,A,B,E(Kas:Na,M,A,B)$
(4) $Z(B) \rightarrow A\ :\ M,E(Kas:Na,M,A,B)$

Z is a dishonest agent controlled by the intruder. In this attack, message (1) from A must be detained from the destined receiver B so that Z can successfully disguise B to send message (4) and complete this session of the protocol. Otherwise, if message (1) wasn't detained, then it would eventually achieve B, and B's answer would also achieve A, which means that the above attack couldn't success.

In [4], detaining is not successfully considered, original messages are not detained, and will successfully achieve destined received, as well as faked messages. Our model involves both intercepting and detaining successfully. The intruder has nondeterministic choice of actions on messages: intercept or detain. Technically, we introduce strategy vectors for the intruder which help him make decisions.

Definition 1. $SV_{\mathcal{P}}(s)[1..n]$ is a **strategy vector** for the intruder, where $\mathcal{P}$ is a protocol, s is a session number, and n is the number of the steps in $\mathcal{P}$. $\mathrm{SV}_{\mathcal{P}}(s)[i]$ assumes values in $\{0,1\}$. $SV_{\mathcal{P}}(s)[i]$=1 means that the intruder intercepts the ith message in session s, and 0 means that the intruder detains it.

Thus strategy vectors can be used in the preconditions of Horn clauses, directing the traces, which will be expatiated in the following subsection.

3) Protocol Dependent Clause.: The content of protocol dependent clauses depends on the concrete protocol. As an example, we choose SPLICE/AS authentication protocol[3].

In any trace t, a new session of a protocol can be started with a fresh session number s. We use an agent variable to correspond each role, except the constant servers such as *AS* in SPLICE/AS authentication protocol. When a new session begins, the agent variables come to a starting state: ready for message sending and receiving. We use the following

clause to represent the starting of a session of SPLICE/AS authentication protocol, which means that all the participants get ready for sending and receiving messages:

$Fresh(t,s) \rightarrow T([ST(C,1,1,<C,S,AS>),s]$–$[ST(S,2,1,$ $<C,S,AS>),s]$–$[ST(AS,3,1,<C,S,AS>),s]$–$t)$

Then C can send the first message, while the constant server *AS* is ready to receive it. The clause for C sending the first message is as follows:

$In([ST(C,1,1,<C,S,AS>),s],t), Notplayed(C,2,s,t) \rightarrow$ $T([<C,S,N_1(C,S,AS,s>),s]$–$[ST(C,1,2,<C,S,AS,$ $N_1(C,S,AS,s)>),s]$–$t)$

Now we discuss how to represent *AS* receiving the first message and then to send the second message. There is a predicate $Accept(a,m,s,t)$ to help the receiver a check if the message m is admissible and decide whether to admit it in the session s in the trace t or not. If $Accept(a,m,s,t)$ holds, then the agent a receive m, else a reject it.

The intruder in the environment, who has a strategy vector $SV_{\mathcal{P}}(s)[\ ]$, can interfere with the receiving of messages. $SV_{\mathcal{P}}(s)[\ ]$ is instantiated when each session begins. The instantiated value of $SV_{\mathcal{P}}(s)[i]$ determines whether the intruder detains the ith message $m_i(x_1,..,x_{N_{\mathcal{P}}},s)$ of the protocol from the receiver or not. Since any message can be received if and only if it can be known by the intruder and the value of corresponding element of the strategy vector equal to 1, we use $I(m_i(x_1,..,x_{N_{\mathcal{P}}},s))\wedge = (SV_{\mathcal{P}}(s)[i],1) \wedge Accept(x_k,m_i(x_1,..,x_{N_{\mathcal{P}}},s),s,t)$ to represent the reception of the message $m_i(x_1,..,x_{N_{\mathcal{P}}},s)$, where i denotes $m_i(x_1,..,x_{N_{\mathcal{P}}},s)$ is the ith message in the protocol.

Note that the intruder can also fake messages relying on its knowledge and computing ability. Hence for each of the reception there is another accompanying clause, with $SV_{\mathcal{P}}(s)[i]$=0. for simplicity, We denote the faked message of $m_i(x_1,..,x_{N_{\mathcal{P}}},s)$ as $m_i(x_1,..,x_{N_{\mathcal{P}}},s)'$.

Hence each following step of the rest of the session has two clauses, one is for $SV_{\mathcal{P}}(s)[i]$=0, the other for $SV_{\mathcal{P}}(s)[i]$=1. As for SPLICE/AS authentication protocol, the reception of the first message which leads to the sending of the second message, is as follows:
$I(N_1(C,S,AS,s),t),\ =(SV_{\mathcal{P}}(s)[1],1), In([ST(AS,3,1,<$ $C,S,AS\ >),s],t), Notplayed(AS,2,s,t), v\ \rightarrow\ T([u,s]-$ $[ST(AS,3,2,\ <C,S,AS,N_1(C,S,AS,s),u>),s]$–$t)$

and
$I(N_1(C,S,AS,s),t),\ =(SV_{\mathcal{P}}(s)[1],0), In([ST(AS,3,1,<$ $C,S,AS>),s],t), Notplayed(AS,2,s,t), v'\ \rightarrow\ T([u',s]-$ $[ST(AS,3,2,<C,S,AS,N_1(C,S,AS,s)',u'>),s]$–$t)$
where
u=$<AS,\{AS,C,N_1(C,S,AS,s),K_s\}_{prv(AS)}>$
u'=$<AS,\{AS,C,N_1(C,S,AS,s)',K_s\}_{prv(AS)}>$
v=$Accept(AS,N_1(C,S,AS,s),s,t)$
v'=$Accept(AS,N_1(C,S,AS,s)',s,t)$

For space limitation, we don't give all the clauses representing the protocol.

C. Representation of Security Properties

This section focuses on security properties of protocols. We consider secrecy property and authentication property, which can be expressed as trace properties. If a property can also be formalized as a Horn clause, then we can put the clause with the Horn logic program for the protocol together and work on their common semantics. For clarity, we name the clauses representing protocols as **protocol clause**, and the clauses representing properties as **property clause**.

In each property clause, there are certainly definite number of honest agent variables who are pairwise different, which we will expatiate in Section IV-B. Let the number of these variables be n_φ.

We can see that there is no dishonest agent variables in property clauses because dishonest agents are already modelled in protocol clauses and property clauses only assert relations between honest agents.

Definition 2. Secrecy property: *In any trace t and any session s, if* $x_1,..,x_{n_\varphi}$ *are pairwise different honest agents, then the secrecy* $m(x_1,..,x_{n_\varphi},s)$ shared by $x_1,..,x_{n_\varphi}$ *can't be known by the intruder.*

In Horn clause form, it will be the following clause($1 \leq i,j, \leq n_\varphi$ and $i \neq j$):
$(\forall x_1,..,x_{n_\varphi},t,s)...,\neq(x_i,x_j),..,Ha(x_1),..,Ha(x_{n_\varphi}),T(t),$ $I(m(x_1,..,x_{n_\varphi},s),t) \rightarrow$

Definition 3. Authentication property: *In any trace t and any session s, whenever the honest authenticator* x_a *receives the final authenticating message* $m_k(x_1,..,x_{n_\varphi},s)$*, the honest authenticatee* x_b *has already sent it in this session and this trace.* $1 \leq a,b \leq n_\varphi$.

In Horn clause form, it will be the following clause($1 \leq i,j, \leq n_\varphi$ and $i \neq j$):
$(\forall x_1,..,x_{n_\varphi},t,s)... \neq(x_i,x_j),..,Ha(x_1),..,Ha(x_{n_\varphi}),T(t),$ $I(m_k(x_1,..,x_{n_\varphi},s),t),(SV_{\mathcal{P}}(s)[k],1), Accept(x_a,m_k(x_1,$ $..,x_{n_\varphi},s),s,t)\ \rightarrow\ In([ST(x_b,r,k,<..,m_k(x_1,..,x_{n_\varphi},s)>$ $),s],t)$

As we have done in Section II-B, we will also use $I(m_k(x_1,..,x_{n_\varphi},s))\wedge =(SV_{\mathcal{P}}(s)[k],1) \wedge Accept\ (x_i,m_k(x_1,$ $..,x_{n_\varphi},s),s,t)$ to represent the reception of $m_k(x_1,..,\ x_{n_\varphi},s)$ in the above clause. The authenticating message $m_k(x_1,..,$ $x_{n_\varphi},s)$ is parameterized by the session number s, so this description complies with Lowe's definition of injective agreement as authentication property[7].

Let φ be a security property. If φ is a secrecy property, then n_φ is the number of agents in a session who share the secrecy $m(x_1,..,x_{n_\varphi},s)$. If φ is an authentication property, then n_φ is the number of agents as parameters in the authenticating

message $m_k(x_1,..,x_{n_\varphi},s)$ which the authenticator and the authenticatee agree on in a session. In both cases, $n_\varphi \leq N_\mathcal{P}$.

We can see that there needn't be any agent variables other than the honest agent variables $x_1,..,x_{n_\varphi}$ in property clauses.

It's obvious that other security properties can be easily considered if they can also be expressed in horn clauses as trace properties.

III. Semantical Relationship: Protocol & Property

Now that the protocols and properties have been formalized, the subsequent problem comes: Does a protocol $\mathcal{P}$ satisfy a property φ? I.e., do their semantics agree with each other?

For convenience, we denote the Horn logic program for protocol $\mathcal{P}$ as $C_\mathcal{P}$, the set of protocol independent clauses as C_I, and the set of protocol dependent clauses as C_D, the clause for property φ as C_φ. Then $C_\mathcal{P}$=$C_I \cup C_D$.

It would be very complicated to deal with random semantics of logic programs directly. Since a logic program has a model if and only if it has a Herbrand model[1], we consider Herbrand models of $C_\mathcal{P}$.

All the symbols in C_φ also occur in $C_\mathcal{P}$, which implies that C_φ is actually expressed in the underlying first order language for $C_\mathcal{P}$, So C_φ has a definite truth value under any interpretation and any assignment for $C_\mathcal{P}$. According to the truth value relationship between $C_\mathcal{P}$ and C_φ under interpretations, we can judge whether a protocol $\mathcal{P}$ satisfies a property φ.

Definition 4. *A protocol $\mathcal{P}$* **satisfies** *a property φ, if C_φ is true under any Herbrand model H of $C_\mathcal{P}$*, i.e., $C_\mathcal{P} \models C_\varphi$.

Definition 5. *A protocol $\mathcal{P}$* **dissatisfies** *a property φ, if C_φ is false under any Herbrand model H of $C_\mathcal{P}$*, i.e., $C_\mathcal{P} \not\models_H C_\varphi$.

Definition 6. *Let θ be an assignment under a Herbrand model H of $C_\mathcal{P}$, and $GC_\mathcal{P}$ be the set of all ground instances of the clauses in $C_\mathcal{P}$, $gc \in GC_\mathcal{P}$,if there is an element b in the Herbrand base of $C_\mathcal{P}$such that $\theta(x)$=b and b is an instance of x in gc, then θ* **takes place in** *gc.*

If $C_\mathcal{P} \not\models_H C_\varphi$, then there exists an assignment θ_φ under H, and a subset of ground instances of the clauses in $C_\mathcal{P}$, denoted as $GC_{\mathcal{P},\theta_\varphi}$, which contains exactly all the ground clause instances in which the assignment θ_φ takes place, such that the truth value of $\theta_\varphi(C_\varphi)$ is *false*, while the truth value of each ground clause in $GC_{\mathcal{P},\theta_\varphi}$ is *true*.

We denote $C_\mathcal{P}$'s least Herbrand model as $H_\mathcal{P}$, the set of logical consequences of $GC_{\mathcal{P},\theta_\varphi}$ as $H_{\mathcal{P},\theta_\varphi}$.

Lemma 1. $H_{\mathcal{P},\theta_\varphi} \subseteq H_\mathcal{P}$.

Proof: It's well known that the set of logical consequences of a logic program is equal to its least Herbrand model[1]. Since $GC_{\mathcal{P},\theta_\varphi}$ is a subset of all the ground instances of the clauses in $C_\mathcal{P}$, then the logical consequences of $GC_{\mathcal{P},\theta_\varphi}$ is also a subset of logical consequences of $C_\mathcal{P}$, that is, $H_{\mathcal{P},\theta_\varphi} \subseteq H_\mathcal{P}$. ■

Now let's extend the meaning of the symbol $\not\models$ in order that it can be applied between not only closed clauses but also ground clauses.

Lemma 2. If $H_{\mathcal{P},\theta_\varphi} \not\models \theta_\varphi(C_\varphi)$ then $H_\mathcal{P} \not\models \theta_\varphi(C_\varphi)$.

Proof: By Lemma 1, $H_{\mathcal{P},\theta_\varphi} \subseteq H_\mathcal{P}$, let $H'_\mathcal{P}$=$H_\mathcal{P} - H_{\mathcal{P},\theta_\varphi}$. $H_{\mathcal{P},\theta_\varphi}$ is the set of logical consequences of $GC_{\mathcal{P},\theta_\varphi}$ who contains exactly all the ground clause instances in which the assignment θ_φ takes place. So θ_φ doesn't occur in $H'_\mathcal{P}$, that is, $H'_\mathcal{P}$ has no impact on the truth value of $\theta_\varphi(C_\varphi)$, hence $H_\mathcal{P} \not\models \theta_\varphi(C_\varphi)$. ■

Lemma 3. If a protocol $\mathcal{P}$ dissatisfies a property φ, then $\not\models C_\varphi$.

Proof:

$C_\mathcal{P} \not\models_H C_\varphi$

$\Rightarrow C_\mathcal{P} \not\models_H \theta_\varphi(C_\varphi)$

$\Rightarrow GC_{\mathcal{P},\theta_\varphi} \not\models_H \theta_\varphi(C_\varphi)$

$\Rightarrow H_{\mathcal{P},\theta_\varphi} \not\models_H \theta_\varphi(C_\varphi)$

$\Rightarrow H_\mathcal{P} \not\models_H \theta_\varphi(C_\varphi)$, by Lemma 2

$\Rightarrow H_\mathcal{P} \not\models_{H_\mathcal{P}} \theta_\varphi(C_\varphi)$, since $H_\mathcal{P} \subseteq H$

$\Rightarrow H_\mathcal{P} \not\models_{H_\mathcal{P}} C_\varphi$

$\Rightarrow \not\models_{H_\mathcal{P}} C_\varphi$. ■

Definition 7. If $GC_{\mathcal{P},\theta_\varphi} \not\models_H \theta_\varphi(C_\varphi)$, then $GC_{\mathcal{P},\theta_\varphi}$ is called an **attack** on $\mathcal{P}$ for φ.

Lemma 4. There is an attack on a protocol $\mathcal{P}$ for a property φ if $\not\models_{H_\mathcal{P}} C_\varphi$.

Proof: $\not\models_{H_\mathcal{P}} C_\varphi$, $H_\mathcal{P}$ is the least Herbrand mode of $C_\mathcal{P}$, so $C_\mathcal{P} \not\models_H C_\varphi$, then $\mathcal{P}$ dissatisfies φ, so there exist an assignment θ_φ under $H_\mathcal{P}$ such that $GC_{\mathcal{P},\theta_\varphi} \not\models_H \theta_\varphi(C_\varphi)$. By Definition 7, there is an attack on $\mathcal{P}$ for φ. ■

Theorem 1. There is an attack on a protocol $\mathcal{P}$ for a property φ iff $\not\models_{H_\mathcal{P}} C_\varphi$.

Proof: It is proved by Lemma 3 and Lemma 4. ■

IV. Using Projection to Limit Agent Number in Attacks

With the semantical relationship discussed, the problem that how to automatically check whether a protocol satisfies properties follows naturally, as done by Blanchet[5][6]. We consider another important problem: when analyzing a protocol for some property, how many agents should we consider?

A. The Infinity of Agent Numbers

When there is an attack $GC_{\mathcal{P},\theta_\varphi}$ on $\mathcal{P}$ for φ, we can't be sure about how many agents are assigned to the roles(i.e.,

agent variables) in $C_{\mathcal{P}}$ to get this attack. $GC_{\mathcal{P},\theta_\varphi}$ contains exactly all the ground clause instances in which the assignment θ_φ takes place, but these ground clause instances may also contain other assignments. Hence there seems to be infinite agents to express an attack, and maybe we have to consider infinite agents in analyzing protocols. Fortunately, Comon has introduced a technique of projection over agents[4]. We have improved this technique and clarified the supremum of the number of agent needed.

B. Roles vs Agents

At run-time, roles(agent variables) are instantiated by agents. It's very important whether two different roles in a protocol $\mathcal{P}$ can be instantiated by a same agent, that is, whether an agent can participate in a protocol session as 2 roles. We are in accordance with Even's viewpoint that if a protocol is designed for 3 parties it should be played by 3 distinct users and not by 2 users[9]!

As to dishonest agents, it depends on whether the security property being analyzed is a strong or weak[9]. A security property is **strong** means that in a protocol session agents are mutually different when this property is being analyzed, while **weak** means that only honest agent are mutually different.

C. The Projection

We make a projection on the Herbrand universe and Herbrand base of $C_{\mathcal{P}}$. Let $GC_{\mathcal{P},\theta_\varphi}$ be the attack on $\mathcal{P}$ for φ. There is definite number of distinct agents in $\theta_\varphi(C_\varphi)$, which is n_φ according to Section II-C. Let the honest agents in $\theta_\varphi(C_\varphi)$ be $S_h^{m_1}(h)$,.., $S_h^{m_{n_\varphi}}(h)$, $0 \le m_i < m_j$ if $i < j$.

In $GC_{\mathcal{P},\theta_\varphi}$, except for $S_h^{m_1}(h)$,.., $S_h^{m_{n_\varphi}}(h)$, there are still other honest agents and dishonest agents. Let's build a projection function f over the Herbrand base:

$$\begin{cases} f(S_h^{m_i}(h)) = S_h^{i-1}(h), \\ \quad for\ i = 1,..,n_\varphi \\ f(S_h^k(h)) = S_d^{k_f}(d), \\ \quad for\ k \neq m_i,\ i = 1,..,n_\varphi, S_d^{k_f}(d)\ doesn't\ occur\ in\ GC_{\mathcal{P},\theta_\varphi} \\ f(u(t_1,..,t_n)) = u(f(t_1),..,f(t_n)), \\ \quad if\ term\ u() \in \textbf{Server} \cup \textbf{Da}, or\ u() \notin \textbf{Agent} \\ f(A(t_1,..,t_n)) = A(f(t_1),..,f(t_n)), \\ \quad if\ the\ predicate\ A(t_1,..,t_n) \neq Ha(S_h^k(h)) \\ f(Ha(S_h^k(h))) = Da(S_d^{k_f}(d)) \end{cases}$$

The purpose of $f(Ha(S_h^k(h)))$=$Da(S_d^{k_f}(d))$ is to keep the truth value of sort assertion predicates under *f*. Under this projection, dishonest agents and servers are projected to themselves, honest agents in $\theta_\varphi(C_\varphi)$ are projected to the fixed agents: $S_h^0(h)$,..,$S_h^{n_\varphi - 1}(h)$, honest agents not in $\theta_\varphi(C_\varphi)$ are projected to some new dishonest agents. Let $f(GC_{\mathcal{P},\theta_\varphi})$ be the set resulted from applying f on each ground clause of $GC_{\mathcal{P},\theta_\varphi}$. There are only n_φ honest agents in $f(GC_{\mathcal{P},\theta_\varphi})$. If we can prove that $f(GC_{\mathcal{P},\theta_\varphi})$ is also an attack, then we can assert that we have determined the number of honest agents in attacks on $\mathcal{P}$ for φ.

D. Semantical Preservation

Definition 8. A ground predicate c is **positively preserved** *by a projection p*: *if the truth value of c is true, then the truth value of* $p(c)$ *is also true*. *Additionally*, *a ground predicate set is* **positively preserved** *by p if all ground predicates in it are positively preserved by p.*

Lemma 5. If $\mathcal{L} \in H_{\mathcal{P}}$ is a ground instance of the postcondition of an unit clause (a clause that has no precondition) in $C_{\mathcal{P}}$, then $\mathcal{L}$ is positively preserved by *f*.

Proof: $\mathcal{L} \in H_{\mathcal{P}}$ is a ground predicate, then there is an unit clause $c_{\mathcal{L}} \in C_{\mathcal{P}}$ such that $\mathcal{L}$ is a ground instance of $c_{\mathcal{L}}$'s postcondition.

If $c_{\mathcal{L}}$ is universally quantified, then $f(\mathcal{L})$ is also a ground instance of $c_{\mathcal{L}}$ as $\mathcal{L}$, hence the truth value of is also *true*.

If $c_{\mathcal{L}}$ is itself a ground unit clause, then $c_{\mathcal{L}} = \rightarrow \mathcal{L}$. The postconditions of ground unit clauses in $C_{\mathcal{P}}$ are sort assertions, $\neq()$ and $=()$. According to the definition of *f*, the truth value of $f(c_{\mathcal{L}})$ agree with that of $c_{\mathcal{L}}$.

Hence by Definition 8, $c_{\mathcal{L}}$ is positively preserved by *f*. ■

Lemma 6. The least Herbrand model $H_{\mathcal{P}}$ of $C_{\mathcal{P}}$ is positively preserved by *f*.

Proof: The least Herbrand model $H_{\mathcal{P}}$ of $C_{\mathcal{P}}$ is equal to the set of logical consequences of the corresponding logic program $C_{\mathcal{P}}$. For each predicate $\mathcal{L}$ in $H_{\mathcal{P}}$, there is a minimal deducting sequence to get $\mathcal{L}$ from $C_{\mathcal{P}}$. Let the length of the minimal deducting sequence be $n_{\mathcal{L}}$, then we can make induction on $n_{\mathcal{L}}$ to prove that $\mathcal{L}$ is positively preserved by *f*.

First, if $n_{\mathcal{L}}$=1, then $\mathcal{L}$ is a ground instance of the postcondition of an unit clause in $C_{\mathcal{P}}$, by Lemma 5, $\mathcal{L}$ is positively preserved by *f*.

Assume $k \ge 1$ and that $\mathcal{L}$ is positively preserved by f for $n_{\mathcal{L}} \le k$.

If $n_{\mathcal{L}} = k$+1, then there must be a ground clause $c_{\mathcal{L}} = \forall(..).A_1, A_2, ..A_m \rightarrow B$ and an assignment σ such that $\mathcal{L} = \sigma(B)$, and $\sigma(A_i) \in H_{\mathcal{P}}$. Since $n_{\sigma(A_i)} \le k$, by the induction, $\sigma(A_i)$ are all positively preserved by *f*. From the definition of *f*, we can infer that $f(\sigma(A_i)) = \sigma_f(f(A_i))$. According to Section II-C, *Ha*() only occurs in $c_{\mathcal{L}}$ that is an unit clause, so A_i and B are not of the form of *Ha*(). Thus $f(\sigma(A_i)) = \sigma_f(A_i)$ and $\sigma_f(A_i) \in H_{\mathcal{P}}$. Now that $\sigma_f(A_i) \in H_{\mathcal{P}}$ and $c_{\mathcal{L}} \in C_{\mathcal{P}}$, then $\sigma_f(f(B)) = \sigma_f(B) \in H_{\mathcal{P}}$, so the truth value of $f(\mathcal{L})$ agrees with that of $\mathcal{L}$. Then $\mathcal{L}$ is positively preserved by *f*.

Moreover, $H_{\mathcal{P}}$ is positively preserved by *f*. ■

We extend f a little more such that it can be applied to not only Herbrand base but also ground clauses. If a ground clause $c = A_1, .., A_m \rightarrow B$, then $f(c) = f(A_1), .., f(A_m) \rightarrow f(B)$.

Definition 9. *A ground predicate c is* **negatively preserved** *by a projection p*: *if the truth value of c is false, then the truth value of* $p(c)$ *is also false.*

In Section II-C we have introduced the secrecy property and authentication property and the Horn clauses for them. Now

we concern the truth value of the ground Horn clause for these properties under the projection *f*.

Lemma 7. If $\theta_\varphi(C_\varphi)$ is a ground Horn clause for a secrecy property φ, then $\theta_\varphi(C_\varphi)$ is negatively preserved by *f*.

Proof: Since φ is a secrecy property, $\theta_\varphi(C_\varphi)$ must be of the form $\neg A_1 \vee, .., \vee \neg A_m$, which is equivalent to $\neg(A_1 \wedge, .., \wedge A_m)$.

If the truth value of $\theta_\varphi(C_\varphi)$ is *false*, then the truth value of $A_1, .., A_m$ are all *true*, so $A_1, .., A_m$ are all logical consequences of $C_\mathcal{P}$, which means $A_1, .., A_m \in H_\mathcal{P}$. By Lemma 6, $A_1, .., A_m$ are all positively preserved by *f*, then the truth value of $f(A_1), .., f(A_m)$ are all *true*, and $f(\neg A_1 \vee, .., \vee \neg A_m)$ is *false*. Hence according to definition 9, $\theta_\varphi(C_\varphi)$ is negatively preserved by *f*. ■

Lemma 8. If e is a ground term of sort **Event**, s is a session number, and t is a ground term of sort **Trace**, then the ground predicate $In([e,s], t)$ is both positively and negatively preserved by *f*.

Proof: If the truth value of *In*([e,s], t) is *true*, then $t = t_1$–[e,s]–t_2, and f(*In*([e,s], t))=*In*([$f(e)$,$f(s)$], $f(t)$)=*In*([$f(e)$,s], $f(t_1)$–[$f(e)$,s]–$f(t_2)$). Hence the truth value of f(*In*([e,s], t)) is also *true*, that is, *In*([e,s], t) is positively preserved by *f*.

On the other hand, If the truth value of *In*([e,s], t) is *false*, then [e,s] doesn't occur in t. By the reduction to absurdity, assume the truth value of f(*In*([e,s], t)) is *true*, then f(*In*([e,s], t)) = *In*([$f(e)$,s], $f(t_1)$–[$f(e')$,s]–$f(t_2)$) and $f(e)$=$f(e')$. By the definition, f is an injective projection. So e=e'. Thus [e,s] occurs in t, which leads to a contradiction.

So *In*([e,s], t) is both positively and negatively preserved by *f*. ■

Lemma 9. If $\theta_\varphi(C_\varphi)$ is a ground Horn clause for authentication property φ, then $\theta_\varphi(C_\varphi)$ is negatively preserved by *f*.

Proof: Since φ is an authentication property, $\theta_\varphi(C_\varphi)$ must be of the form $A_1, .., A_m \rightarrow B$, and B is of the form *In*([e,s], t), where $e$and t are both ground terms. By Lemma 8, *In*([e,s], t)is both positively and negatively preserved by *f*. Following the ways in Lemma 7, we can infer that $A_1, .., A_m$ are all positively preserved by *f*. So if the truth value of $A_1, .., A_m \rightarrow B$ is *false*, then the truth value of $f(A_1, .., A_m \rightarrow B)$ is also *false*. Hence $\theta_\varphi(C_\varphi)$ is negatively preserved by *f*. ■

Theorem 2. If there is an attack involving arbitrary number of honest agents on a protocol $\mathcal{P}$ for φ, then there is an attack on $\mathcal{P}$ for φ involving at most $N_\mathcal{P}$ honest agents. That is, the supremum of the number of necessary honest agents in analysis of security properties is $N_\mathcal{P}$.

when φ is considered as a strong security property, the number of dishonest agents is open, we can't give a supremum. But by compactness theorem, we can point out that for any attack $f(GC_{\mathcal{P},\theta_\varphi})$, there is a finite ground predicate set $f(GC_{\mathcal{P},\theta_\varphi})' \subseteq f(GC_{\mathcal{P},\theta_\varphi})$ such that $f(GC_{\mathcal{P},\theta_\varphi})' \nvDash \theta_\varphi(C_\varphi)$, that is, any attack can be expressed using finite number of dishonest agents, while not definite.

E. Supremum of Dishonest Number of Agents when φ is Weak

when φ is a weak security property, we can project all agents in $GC_{\mathcal{P},\theta_\varphi}$ not in $\theta_\varphi(C_\varphi)$ to a single dishonest agent, such as *d*. Following the above ways, it's easy to prove that $f(GC_{\mathcal{P},\theta_\varphi})$ is still an attack. Thus in this case, only 1 dishonest is needed. Associating with Theorem 2, we can get the following corollary:

Corollary 1. *If φ is a weak property, then any attack can be expressed with only* 1 *dishonest agent. That is, the supremum of the number of necessary agents in analysis of security properties is $N_\mathcal{P} + 1$.*

V. Conclusion and Future Work

We describe a trace model for protocols and properties based on Horn logic, which adequately covers the ability of the intruder and more attacks.

Strategy vectors ensure that the intruder can have nondeterministic choices, which help to correctly and completely describe the reception of messages. Without this extension, detaining of messages can be successfully modelled, and many attacks would be missed. This model is universal for any protocols that can be described by logic program.

The supremum of the number of honest agents needed to express an attack is given as $N_\mathcal{P}$. Our proof is based on a projection that reduces the number of honest agents in an attack to less than or equal to $N_\mathcal{P}$ while preserving the semantics of the attack. The supremum of the number of dishonest agents needed to express an attack on a protocol $\mathcal{P}$ is proved to be dependent on whether the security property being analyzed is considered as a strong one or a weak one. For strong security properties, the supremum of the number of dishonest agents is 1, otherwise, the supremum is open.

References

[1] Loyd J.W.: Foundations of logic programming. Springer-Verlag, 1987.
[2] Dolev D., Yao A.C.: On the security of public key protocols. IEEE Transactions on Information Theory, 2(29):198–208, 1983.
[3] Clark J., Jacob J.: A survey of authentication protocol literature: Version 1.0. 1997.
[4] Comon H., Cortier V.: Security Properties: Two Agents are Sufficient. Science of Computer Programming 50(1-3), pages 51-71, 2004.
[5] Blanchet, B.: From secrecy to authenticity in security protocols. In 9th International Static Analysis Symposium (SAS'02), pages 242-259, 2002.
[6] Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In 14th IEEE Computer Security Foundations Workshop (CSFW '01), pages 82-96. IEEE, 2001.
[7] Lowe, G.: A hierarchy of authentication specifications. In Proceedings of the 11th. IEEE Computer Security Foundations Workshop, pages 31-43. IEEE Computer. Society Press, 1997.
[8] Even S., Goldreich O.: On the security of multi-party Ping-Pong protocols, Technical Report 285, Technion - Israel Institute of Technology, Computer Science Department, 1983.
[9] http://www.wisdom.weizmann.ac.il/∼oded/PS/eg83rev.ps.
[10] Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. Proc. TACAS, no.1055 in LNCS, Springer-Verlag, 1996.

Towards the Detection of Emulated Environments via Analysis of the Stochastic Nature of System Calls

Tauhida Parveen
Dept. of Computer Sciences
Florida Institute of Technology
tparveen@fit.edu

William Allen
Dept. of Computer Sciences
Florida Institute of Technology
wallen@cs.fit.edu

Scott Tilley
Dept. of Computer Sciences
Florida Institute of Technology
stilley@cs.fit.edu

Gerald Marin
Dept. of Computer Sciences
Florida Institute of Technology
gmarin@cs.fit.edu

Richard Ford
Dept. of Computer Sciences
Florida Institute of Technology
rford@cs.fit.edu

Abstract

One of the most powerful tools in the hacker's reverse engineering arsenal is the virtual machine. These systems provide a simple mechanism for executing code in an environment in which the program can be carefully monitored and controlled, allowing attackers to subvert copy protection and access trade secrets. One of the challenges for anti-reverse engineering tools is how to protect software within such an untrustworthy environment. From the perspective of a running program, detection of the emulated environment is not trivial, as the attacker can emulate the result of different operations with arbitrarily high fidelity. Thus, an emulated environment may be - prima facie - indistinguishable from a "real" environment. However, this conclusion may well be false: this paper demonstrates a mechanism that is able to detect even carefully constructed virtual environments by focusing on the stochastic variation of system call timings. A statistical technique for detecting emulated environments is presented, which uses a model of "normal" system call behavior to successfully identify two commonly used virtual environments under realistic conditions.

Keywords: reverse engineering, security, digital rights management, emulation, virtual machine

1. Introduction

Virtual or emulated execution environments are being applied to a variety of new applications, such as software testing [7], distributing pre-configured software [11], and enhancing computer science education [3]. While such an environment may be used for purely legitimate purposes, the unauthorized or malicious use of these technologies is also increasing. One such use is the reverse engineering of binaries that contain protected or proprietary information. This motivates the need for techniques that can determine programmatically whether an application or operating system is executing in an emulated or virtual environment.

This paper presents a technique that has shown promise in detecting when a program is executing in a virtual environment, without prior knowledge of the specific environment in use. This technique relies on changes in the distribution of system call timing that result from the additional processing time required to provide a non-native execution environment. A model of "normal" system call timing is derived for a range of hosts and that model is used to successfully detect program execution in two common virtual environments.

The next section of this paper briefly describes native and emulated execution environments. Section 3 details the development of the detection methodology and describes a proof-of-concept implementation of the approach. Section 4 outlines experiments that were conducted to test the methodology and provides an evaluation of the results. Section 5 summarizes the paper and discusses possible avenues for future work.

2. Background

Chikofsky & Cross define reverse engineering as "analyzing a subject system to identify its current components and their dependencies, and to extract and create system abstractions and design information" [5]. Applications of reverse engineering include construction

of a new software product or maintenance of legacy applications, deconstructing software for the purpose of teaching, repairing malfunctioning systems, porting software to run on a different operating system, or developing new applications that run in conjunction with the legacy software [2].

Reverse engineering can also be used to reveal and circumvent protection mechanisms, possibly leading to unauthorized access to protected intellectual property or digital content. Because binary reverse engineering frequently requires monitoring instruction execution, tools such as debuggers and emulators are necessary. A debugger modifies the object code so that it can track program state or control program execution. However, anti-debugging techniques have been developed that can detect those modifications and prevent further monitoring of program execution [6].

Most emulators (or a dynamic debugger that provides emulated environments) use a compatibility layer that translates system calls to the native execution environment[1] into system calls for the emulated execution environment[2]. Because of this compatibility layer, the access restrictions present in a program may be by-passed in the emulated environment, thus exposing information that would otherwise be protected. Therefore, individuals wishing to circumvent protection mechanisms often use an emulated environment.

Several suggestions have been made to change the technology used to produce software so that it is more resistant to attempts to circumvent copyright protection schemes [4]. These techniques focus on building preventive measures into the application. They assume that the application is executing where it is expected to run, in a native environment. It is possible that these methods can be by-passed or security measures can be removed when the application is being executed in an emulated execution environment.

In this research, virtual machines were used to create emulated environments. Virtual machines are optimized; they generally execute code faster than emulators. The emulated environments created by virtual machines are usually better in performance than traditional emulators [13]. Two commercial virtual machine products VMware Workstation [12] and Virtual PC 2004 [8] were used in this research to provide consistent emulation of x86-based applications. Although these products can support a range of operating systems, the experiments described here were performed using the Windows XP system.

[1] In this research, the term *native* execution environment describes a configuration in which an operating system is installed on a physical machine and interacts directly with the hardware of that machine to support program execution.

[2] An *emulated* execution environment provides an imitation of a native environment where the operating system actually interacts with the physical machine only through a layer of software (a virtual machine or other emulation program) that is itself running in a native execution environment.

3. Detecting an Emulated Environment

The timing of a process' execution is affected by small fluctuations and variations by the execution environment's behavior. These microscopic fluctuations in performance provide the key behavior used in this research to gain a better understanding of the nature of an execution environment.

To improve security and reliability, application software is restricted from gaining direct access to most system resources. System calls are used to request the operating system for restricted actions such as accessing I/O devices or system memory. During the execution of a system call, interrupts and other unpredictable events can cause timing delays. Thus, if one were to measure the duration of individual system calls executed in a native environment, the timing of those individual calls could be expected to show artifacts that represent the interaction of physical and software processes.

This behavior is different in an emulated execution environment and should show variability in timing that is distinct from the durations measured when executing the same system calls in a native environment. This is due to the addition of the compatibility layer and the presence of other applications running in the same native environment as the emulation. Therefore, the detection methodology developed in this research is based on a statistical analysis of the timing properties of system calls in both environments.

The Windows XP environment provides a number of different methods for measuring the passage of time. While the default low-level timer is the *GetTickCount()* function, Windows XP also provides a high resolution counter that can produce more accurate timing measurements. The *QueryPerformanceCounter()* function [9] can be used to retrieve the current values of a high resolution counter. In this research, timing data from system calls executed in both native and emulated environments was gathered by calling the *QueryPerformanceCounter()* function at the beginning and at the end of a section of a code.

It should be noted that the emulation environment might intentionally alter any internal source of timing information in an attempt to hide its presence. However,

the goal of this research is to determine if an emulated environment impacts the system call timing significantly enough for that characteristic to be used for detection, not to present a foolproof detection mechanism.

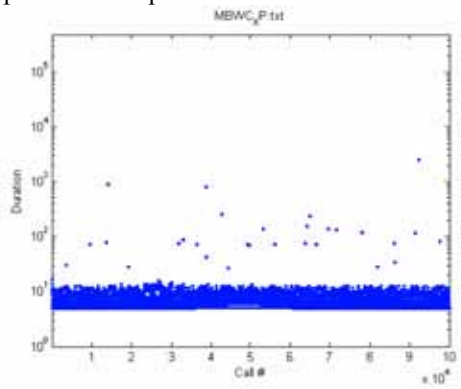


Figure 1 (a): System Calls in Windows XP (log scale)

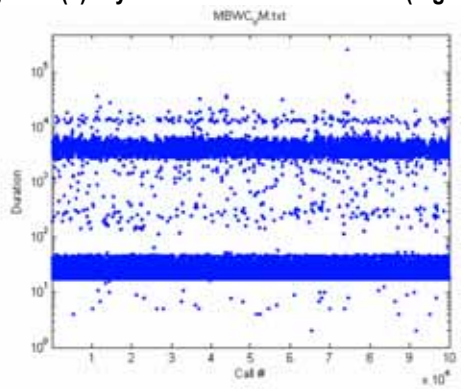


Figure 1 (b): System Calls in VMWare (log scale)

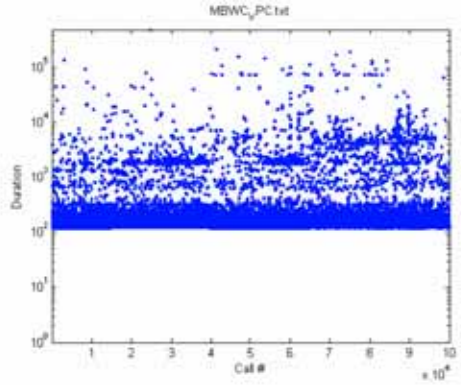


Figure 1 (c): System Calls in Virtual PC (log scale)

The hypothesis that timing artifacts can be used to distinguish between a native and emulated execution environments came from observing the timing variations of system calls from both environments, as shown in Figure 1 (a-c). These plots show the durations (on a log scale) of the same set of system calls executed in each of the three environments (native XP, VMWare and Virtual PC) running on the same platform.

Based on the analysis of this variations in system call timing shown in Figure 1, a methodology was developed that could be used in real-time to determine whether the underlying execution environment was native or emulated. The detection methodology follows three steps which are described in more detail below:

3.1 Gathering System Call Timing Data

The model of normal behavior was created by the execution of system calls in native environments on several different platforms. This information was gathered by a program written in the C language which executes a series of system calls, logging the start and end timestamps for each call. Fifteen non-I/O system calls were chosen from the Win32 API. Table 1 lists the system calls that were used in the experiments. Specific information on individual calls is available from the MSDN Library [10]. System calls that are associated with input/output (I/O) activities were purposely avoided since their timing measurements can be affected by unpredictable hardware or software-related events. Note that this set of system calls is not intended to represent the calls most commonly used in current applications; they were chosen randomly from the non-I/O calls provided by the Win32 API. Further work is needed to determine the best mix of calls to use for reliable detection.

Table 1: The set of Win32 system calls used to create the model of 'normal' behavior

GetVersion	*GetTickCount*
MutiByteToWideChar	*GetComputerName*
IsCharAlpha	*GetSystemTime*
CharLower	*GetTimeZoneInformation*
GetTimeFormat	*CharUpper*
GetSystemInfo	*GetSysColor*
GetSystemDirectory	*GetNumberFormat*
GetLocaleInfo	

Each system call was executed a total of 10,000 times, but the calls were made in a random order so that interactions between calls were not a factor. Each run of the experiment resulted in a total of 150,000 individual calls. The *QueryPerformanceCounter()* function was used to record the start and end timestamps for each system call as they executed in the native execution environment. From the start and end timestamps, the durations of each call were calculated (Duration = end timestamp – start timestamp). The mean and standard deviation of the durations were also calculated.

The durations of the system calls gathered from the native execution environment did not fit any well-known statistical distribution. Therefore, a threshold was determined such that the threshold would, with high probability, contain the system call durations from the native environment but would not contain many of the durations of the system calls executed in non-native execution environments.

3.2 Selecting a Detection Threshold

A number of trials were conducted to determine an optimum threshold that would separate the system call durations gathered from a native execution environment from those gathered from a non-native environment. The range of trial thresholds varied from 1 to 4 standard deviations above the mean of the durations from the native environment. The following steps show how the optimum threshold level was reached.

1. Let T_i be the random variable that is considered to be the duration of the system calls.
2. The sample mean and standard deviation for T_i is calculated.
3. Let C be a constant value from 1 to 4 with an increment of .01
4. Let X_C be the thresholds -
   X_C = *Sample Mean of durations + (1+C * .01)* sample standard deviation*

From the set of thresholds generated by this method, an optimum threshold was chosen such that it contains a significant percentage of the native system call durations with low probability of false alarm. To determine this optimum threshold, the probability that the durations of system calls in the native environment would exceed that threshold level was calculated for each increment in the range of potential threshold values:

1. let P_τ be the probability of system call duration greater than the threshold, τ
2. $$P_\tau = \frac{\text{count the number of duration values that are} > \tau \text{ (threshold)}}{total\_number\_of\_runs}$$

It was found that the probability of durations exceeding 4 standard deviations above the mean was less than 0.05. Therefore, the threshold of four standard deviations above the mean was chosen to distinguish between the native execution environment and the emulated execution environment. (It was determined that this threshold produced an acceptably low false alarm rate, so no calculations were performed above the range of 1 to 4 standard deviations, however it is possible that higher threshold values may produce acceptable results as well.)

However, because the system call durations do not follow a particular distribution there is no guarantee that the probability of durations exceeding the threshold would always be below 0.05. Therefore, Chebyshev's inequality [1] was used to determine the probability of durations that would be allowed to exceed the threshold without generating a false alarm.

Chebyshev's inequality states that in any data sample or probability distribution, nearly all the values are close to the mean value, and provides a quantitative description of terms like *nearly all* and *close to*. For example, no more than 1/4 of the values are more than 2 standard deviations away from the mean, no more than 1/9 are more than 3 standard deviations away and no more than 1/16 are more than 4 standard deviations. Although Chebyshev's theorem states that no more than 1/16 of the data should be *away* from the mean, in this research only the value *above* the mean was considered. Therefore, in a native environment, no more than 1/16 (or 0.0625) of the system call durations will be more than 4 standard deviations above the mean. If measurements determine that the fraction of system calls above this threshold is greater than 0.0625, then a significant environmental change (such as the introduction of an emulator) is likely.

3.3 Measuring System Call Timing Data

The last step for the detection of emulated environments is to compare probabilities of the timing being above the set threshold from the native and the emulated environments.

4. Experiments and Results

Several experiments were carried out to determine if the model created from the system call timing data gathered from native execution environments could be used to reliably determine whether a new execution environment was native or non-native. The model was also tested in a variety of native execution environments to determine if false alarms would occur.

For these experiments, the native environment under study was Windows XP (SP2), installed on a range of machines, each with a different processor speed and mix of installed software. Virtual machine environments, VMware and Virtual PC 2004, were used for these experiments and the operating system used within both of these emulated environments was also Windows XP.

Table 2: Threshold calculated for each processor by the method described in Section 3.2 (columns 2, 3 are based on system call durations measured by *QueryPerformanceCounter()*)

Processor (GHz)	Native Mean	Native Standard deviation	Threshold (Mean + 4*standard deviation)
1.5	24.51	55.79	247.67
1.8	21.71	46.38	207.23
2.26	14.32	29.73	133.24
2.5	14.21	29.16	130.85
3.0	13.58	40.69	176.34
3.4	11.99	35.85	155.39

Table 3: Proportion of system calls exceeding the thresholds shown in Table 2

Processor (GHz)	Threshold from Table 2	Native environment proportion	VMware proportion	Virtual PC proportion
1.5	247.67	0.011133	0.159933	0.304373
1.8	207.23	0.014260	0.167987	0.305567
2.26	133.24	0.008559	0.101880	0.342140
2.5	130.85	0.007613	0.131040	0.113933
3.0	176.34	0.007307	0.176887	0.074800
3.4	155.39	0.009547	0.098053	0.117660

The set of test machines included six PCs, each with a Pentium 4 processor, which were chosen because they provide a representative sample of contemporary hardware. Two of the machines were equipped with a comparatively slow processor (1.5 GHz and 1.8 GHz), two were mid-range machines (2.26 and 2.53 GHz), and the other two are higher performance machines (Hyper-Threaded processors running at 3.0 and 3.4 GHz, respectively). Features such as RAM, disk space, software version, the mix of applications installed on the system, and processor type were evaluated to determine if they would have any impact on the detection methodology, but no evidence was found to support this.

When conducting the experiments to validate the model, the same C program described in Section 3.1 was used to gather timing data from all six sample machines in all three environments (native Windows XP, VMware and Virtual PC). Once the data was gathered from each machine, the technique described in Section 3.2 was used to calculate a threshold for each machine.

Table 2 shows the threshold from each of the six sample machines. The values shown in the second and third columns are calculated from the system call durations measured using *QueryPerformanceCounter()*. The thresholds presented in the fourth column are based on the calculation described in Section 3.2.

In the next step, system call durations gathered from the native and emulated execution environments on each machine were checked against these thresholds. The hypothesis was that most of the system call durations gathered from the native environment (Windows XP) would fall within the threshold. Using Chebyshev's inequality, durations measured in native execution environments should not exceed the threshold by more than 0.0625 (based on the calculations in Section 3.2).

If the proportion of measured durations exceeded the threshold by more than 0.0625, the environment would not fit the model for a native environment and would be classified as an emulated environment.

For each of the execution environments, Table 3 shows the proportion of system call durations that exceeded the corresponding threshold (from Table 2). Figure 2 graphically depicts the information shown in columns three, four and five of Table 3, clearly showing that the proportion is well below 0.0625 (established with Chebyshev's rule) for the native environments and above that value (in many cases significantly above) for each of the emulated environments.

From this observation, it can be deduced that all six native execution environments fit the model and the data from the emulated execution environments do not fit the model. However, some of the emulated environments that were running on high performance processors approach the threshold. To the casual observer, it may appear that a threshold might be found that more evenly separates the native and emulated environments. However, lowering the threshold without knowing the

true distribution of the system call durations risks increasing false alarms.

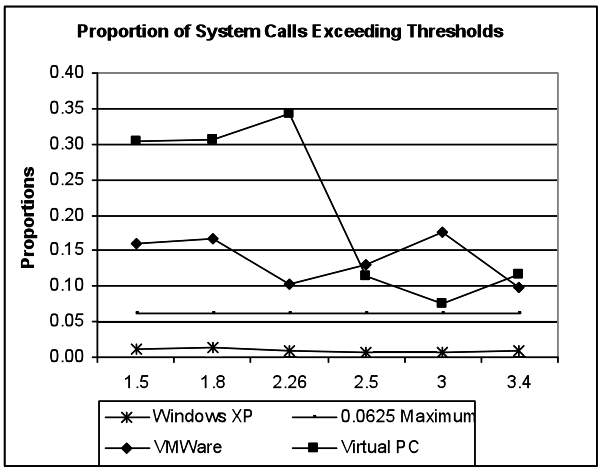


Figure 2: Graphical depiction of the results in Table 3

One solution to this problem would be to use a decision rule for determining when a false alarm occurs. A decision rule could be set such that, if the probability of the durations exceeding the threshold was checked *n* times, then so long as the probability did not exceed the threshold more than *k* out of *n* times, it could still be considered as a native environment.

5. Summary and Future Work

One of the most powerful tools in the hacker's reverse engineering arsenal is the virtual machine. These systems provide a simple mechanism for executing code in an environment in which the program can be carefully monitored and controlled, allowing attackers to subvert copy protection and access trade secrets. Detection of the emulated environment is not trivial, as the attacker can emulate the result of different operations with arbitrarily high fidelity.

This paper presented an approach to countering this threat by providing a means of automatically detecting the attempted reverse engineering. If it can be determined that the execution environment is monitoring the behavior of an application for the purpose of revealing or circumventing protection mechanisms, preventive measures may be taken to protect the application from such threats. The paper also described a proof-of-concept implementation of the approach. An evaluation of the results from experiments conducted to test the methodology suggested that the methodology is sound, and that the approach can successfully detect execution in an emulated environment.

The preliminary research described in this paper also shows that additional work is needed to produce an approach that is refined enough to be of general use. For example, it was mentioned that this approach was developed using non-I/O system calls that were chosen at random. The research could be expanded to include other non-I/O system calls, to determine if some system calls are more useful for building the model than others.

There is also an obvious need to evaluate the approach on non-Windows XP operating systems (and with emulated environments other than VMware and Virtual PC). It would be interesting to try the approach on other platforms, such as Linux, Mac OS X, or even gaming platforms such as the Xbox 360 and with other emulation environments, such as Bochs, Xen or WINE.

References

[1] Arnold A. *Probability, Statistics and Queuing Theory with Computer Science Applications*. Academic Press, 1990.

[2] Bennet, K. and Rajlich, V. "Software Maintenance and Evolution: A Roadmap." In *Proceedings of the 22nd International Conference on Software Engineering* (ICSE 2000). Future of Software Engineering Track, pp 73–87. New York, NY: ACM Press, 2000.

[3] Bullers, W. I.; Burd, S.; and Seazzu, A. "Virtual machines - an idea whose time has returned: application to network, security, and database courses", *Proceedings of the 37th SIGCSE Symposium on Computer Science Education*, 2006

[4] Collberg C. and Thomborson, C. "Watermarking, TamperProofing, and Obfuscation - Tools for Software Protection." *IEEE TSE,* Vol. 28, No. 8, August 2002.

[5] Chikofsy, E.; and Cross, J. "Reverse Engineering and Design Recovery: A Taxonomy." *IEEE Software* 7(1):13-17, January 1990.

[6] Eilam, E. *Reversing: Secrets of Reverse Engineering*, Wiley, 2005

[7] Magnusson, P.S, "The Virtual Test Lab", *IEEE Computer*, Vol. 38, No. 5, pp. 95–97, May 2005

[8] Microsoft Corp. Virtual PC. URL: www.microsoft.com/windows/virtualpc/default.mspx

[9] Microsoft Corp. "How to use QueryPerformanceCounter to Time Code." URL: support.microsoft.com/kb/q172338

[10] Microsoft Corp. MSDN. URL: msdn.microsoft.com

[11] Sapuntzakis, C.; Brumley, D.; Chandra, R.; Zeldovich, N.; et al., "Virtual Appliances for Deploying and Maintaining Software", *Proceedings of the 17th Large Installation Systems Administration Conference (LISA)*, October 2003

[12] VMware Inc., URL: www.vmware.com

[13] Efrem G. Mallach, "On the relationship between virtual machines and emulators", *Workshop on Virtual Computer Systems*, March 1973.

Self-Managed Deployment in a Distributed Environment via Utility Functions

Debzani Deb
Montana State University, Bozeman, MT, USA
debzani@cs.montana.edu

Michael J. Oudshoorn
The University of Texas at Brownsville, TX, USA
michael.oudshoorn@utb.edu

John Paxton
Montana State University, Bozeman, MT, USA
paxton@cs.montana.edu

Abstract

This paper proposes algorithms and mechanisms for achieving self-managed deployment of computationally intensive scientific and engineering applications in highly dynamic and large-scale distributed environment. The primary focus is on the modeling of the application and underlying architecture into a common abstraction and on the incorporations of autonomic features to those abstractions to achieve self-managed deployment. To represent the underlying heterogeneous infrastructure, a hierarchical (tree) model of distributed resources has been adopted that organizes distributed nodes in a utility-aware way. To accomplish the self-adaptive deployment, a utility-function has been formulated that governs both the initial deployment of an application and its dynamic reconfiguration. In our approach, the deployment decisions are made solely based on locally available information and without costly global communication or synchronization. The self-management is therefore decentralized to provide better adaptability, scalability and robustness.

1. Introduction

Many scientific fields, such as genomics, astrophysics, geophysics, computational neuroscience or bioinformatics which require massive computational power and resources, can benefit from a large-scale integrated infrastructure, formed by harnessing the spare compute cycles of distributed computation and communication resources. Typically these applications are composed of a large number of distributed components and it is important to deploy them in the underlying network in a way that meets the computational power and network bandwidth requirements of those components and their interactions. However satisfying these requirements in a large-scale, non-dedicated, heterogeneous, and highly dynamic distributed environment is a significant challenge. As the operating environment and applications grow in scale and complexity, attaining the desired level of performance in this dynamic environment becomes infeasible using current approaches that are based on global knowledge, centralized scheduling and manual reallocation. Therefore, self-managed deployment is paramount to lower operation costs, to manage system complexities and to maximize overall utilization of the system.

This paper proposes algorithms and mechanisms for achieving self-managed deployment of computationally intensive scientific and engineering applications within a highly dynamic distributed environment. The main focus of this paper is to model the application and the underlying architecture into a common abstraction and to incorporate autonomic features [1] to those abstractions to achieve self-managed deployment. To model the underlying heterogeneous infrastructure, we developed techniques that allow the distributed resources to organize in a utility-aware way while assuming minimal knowledge about the system. To achieve self-managed deployment of application components across the network nodes, we designed a scalable and adaptive deployment algorithm that is governed by a utility function [2]. The utility function, which returns the system's overall utility based on different application and system level attributes, governs the initial deployment of the application components and maintains the optimality during their executions despite the dynamism and uncertainty associated with the networked environment. The self-management techniques described in this paper are decentralized and assume minimal knowledge about the environment to provide better adaptability, scalability and robustness.

Fully automating the organization and optimization of a large distributed system is a staggering challenge and there are numerous research groups working toward this goal. Some approaches [3,4] target the development of new autonomic applications to realize the desired benefits of self-management in a distributed environment. In their prototype implementation, Unity [5] achieves self-management via interconnections amongst a number of autonomous agents, however assumes global knowledge in order to optimally allocate the resources in the system. Astrolabe [6] operates by creating a virtual system-wide hierarchical database of the state of a collection of distributed resources, which evolves as the underlying information changes. The AutoFlow [7] project aims to develop a self-adaptive middleware and utilizes a hierarchical organization of underlying resources clustered according to various system attributes for deployment.

The rest of the paper is organized as follows. Section 2 details the design and implementation of different aspects of the proposed application deployment process.

Section 3 presents the experimental evaluation of the proposed deployment and Section 4 concludes the paper.

2. Self-Managed Deployment

As the application components within an application execute with different constraints and requirements, they should be mapped to appropriate hardware resources in the distributed environment so that their constraints are satisfied and they provide the desired level of performance. Mapping between these resource requirements and the specific resources that are used to host the application is not straightforward.

In this paper, a three step process is designed to perform this mapping as shown in Figure 1. In the first step, an application model is extracted that represent an application in terms of its components and their internal dependencies along with the estimated resource requirements of the components and their links. The next step involves constructing a model of the underlying network by obtaining knowledge about available resources such as their computational capabilities, connectivities and workloads and then organizing them according to network proximity. The third and final step allocates a specific set of resources to each application with respect to the resources required by the application components and the resources available in the system. The goal of the mapping is to maximize the system's overall utility based on certain policies, priorities, user-defined constraints and environmental conditions. The important aspects of this deployment process are detailed in the following few sections.

2.1. The Application Model

In this paper, an application is modeled as a graph consisting of application components and the interactions among them. Analyzing and representing software in terms of its components and their internal dependencies is important in order to provide the self-managing capabilities because this is actually the system's view of the run-time structure of a program. Well structured graph-based modeling of an application also makes it easier to incorporate autonomic features into each of the application components.

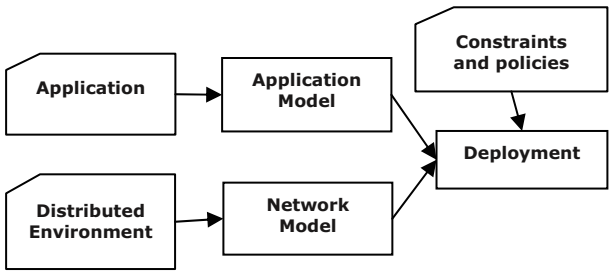


Figure 1. Application deployment process

An application is represented as a node-weighted, edge-weighted directed graph $G = (V,E,w_g,c_g)$, where each vertex $v \in V$ represents an application component and the edge $(u,v) \in E$ resembles the communication from component u to component v. The computational weight of a vertex v is $w_g(v)$ and represents the amount of computation that takes place at component v. The communication weight $c_g(u,v)$ captures the amount of communication (volume of data transferred) between components u and v. The detailed process of the extraction of the graph form of an application is out of the scope of this paper. However, Reference [8] provides a detailed description of our static analysis based application graph construction approach.

2.2. The Network Model

In this research, the target environment for the deployment of the application is a distributed environment consisting of a non-dedicated heterogeneous and distributed collection of nodes connected by a network. To organize the computation around this heterogeneous and distributed pool of resources, traditional approaches rely on the assumption that sufficiently detailed and up-to-date knowledge of the underlying resources is available to a central entity. While this approach results in the optimized utilization of the resources, it does not scale to a large numbers of nodes. Maintaining a global view of a large-scale distributed environment becomes prohibitively expensive, even impossible at a certain stage, considering the unprecedented number of nodes and the unpredictability associated with a large-scale computing system due to various dynamic factors.

We propose a different approach that addresses the above problems and allows the heterogeneous pool of resources to be organized in a structure that facilitates their effective use. The aim is to organize the distributed resources in a structure such that nodes that are *closer* to each other in the structure are also *closer* to each other considering network distance (latency, bandwidth, etc.). Once structured in this way, it is possible to detect higher utility paths locally that correspond to low latency and high bandwidth between network nodes. As a result of that, the deployment of the application graph can be performed in a utility-aware way, without having full knowledge about the underlying resources and without calculating the utility between all pairs of network nodes.

The proposed organization is obtained by modeling the target distributed environment as a tree in which the nodes correspond to compute resources, edges correspond to network connections and execution starts at the root. More specifically, a tree structured overlay network [9,10] is used to model the underlying resources, which is built on the fly on top of the existing network topology.

The important aspect of our design is the emergence of the tree topology, which structures the distributed nodes, in a utility-aware way while assuming minimal knowledge about the environment. Each parent monitors only a limited number of nodes and the deployment decision is made based on this locally available monitored data. The design is therefore suitable for dynamic and large-scale computing environment. Also this model allows us to limit the utility evaluation within a subtree performed by the parent of that subtree, instead of performing the costly utility evaluation globally to determine the highest utility node. Figure 2(a) shows a small computing environment distributed in three domains and Figure 2(b) illustrates a tree overlay network that is built on top of this physical topology.

Formally, the entire network is represented as a weighted tree $T = (N,L,w_t,c_t)$, where N represents the set of computational nodes and L represents network links among them. The computational weight $w_t(n)$ indicates the cost associated with each unit of computation at node n. The communication weight $c_t(m,n)$ models the cost associated with each unit of communication of the link between parent m and child n. When two nodes are not connected directly, their communication weight is the sum of the link weights on the path via their predecessors or successors. Therefore, larger values of node and edge weights translate to slower nodes and slower communication respectively.

To construct an overlay tree, each node is assumed to have a *children list* signifying the URLs of its neighbors that have direct connection with it. The problem of how to generate this list is out of the scope of this research, however it can be addressed by using several tools [11,12]. Once a user starts an application in his/her machine, the graph representation is extracted from the application code. The initiator node then decides which components to execute and which ones to delegate to its best utility child nodes considering all the nodes listed in its children list. The delegated nodes again spread the computation in this manner. The topology of the resulting overlay network thus becomes a tree with the originating machine at the root node.

2.3. The Utility Function

In this research, both the initial placement of the application components and their reconfigurations are governed by utilizing utility functions. Several applications and environment specific attributes are combined in a single utility function. This multi-attribute function returns a scalar value signifying system's overall utility for each possible state of a system and the goal becomes to select a state that maximizes the overall utility. During execution, resource allocation and other operating conditions may change; the corresponding change in the overall utility can be calculated by this utility function and reconfiguration decisions can be taken toward maximizing this value. As computing environments are becoming increasingly large, distributed, complex and dynamic in nature, the optimal actions are likely to evolve over time and a utility function that continuously computes the most desired state is expected to be more suitable in such cases.

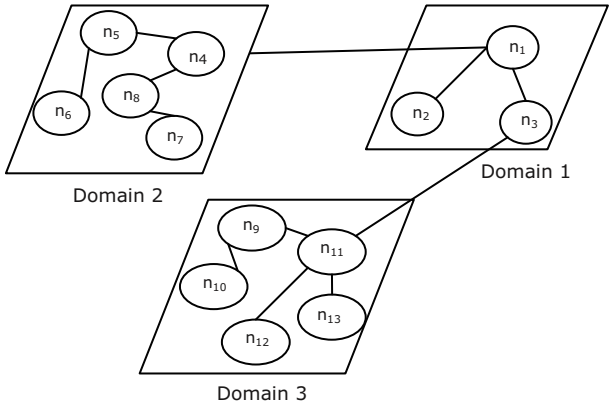


Figure 2(a). Sample distributed environment.

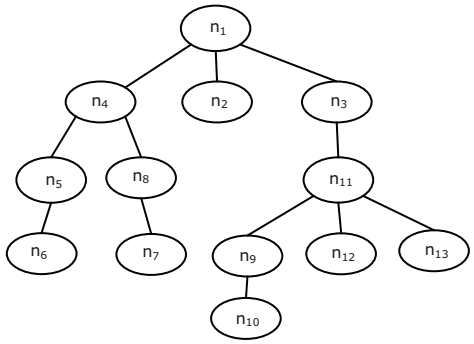


Figure 2(b). Overlay Tree.

In this paper, the utility function is designed to respect the following application, environment and user specific high-level policies:

1. While mapping partitions containing a large number of application components in the tree network, nodes that lead to a wider subtree (higher degree of connectivity) are preferred as higher degree allows more directions for partition growth.
1. Faster and less busy nodes are favored over slower and overloaded nodes when assigning components to resources.
2. Nodes with faster communication links are preferred over nodes with slower communication links.
3. High priority applications are preferred over low priority jobs.

2.4. Initial Deployment

Once the application and underlying resources have both been modeled, the deployment problem reduces to the mapping of different application components and their interconnections to different nodes in the target environment and network links among them so that all requirements and constraints are satisfied and system's overall utility is maximized. The assumption is that the application can be submitted to any node, which acts as the root or starting point of the application. Furthermore the application may end its execution either at the root node or at one or more clients at different destination nodes.

When the application graph G is submitted to the root node of the tree network, the root then decides which application components to execute itself and which components to forward to its child's sub-tree. The child, who has been delegated a set of components again deploys them in the same way to its subtrees. For effective delegation of components at a particular node having |P| children, graph coarsening techniques [13] are exploited to collapse several application components into a single partition, so that ≤ |P| partitions are generated at that stage. The coarsened graph is projected back to the original or to a more refined graph once it is delegated to a child node.

In the above approach, each parent selects the highest utility child to delegate a particular partition (set of components). Finding the highest utility child to delegate a partition to means finding the highest utility mapping M of the edges (v_j,v_k) where $v_j \in V_r$ (represents the set of components that the parent decided to execute itself) and $v_k \in V_s$ (represents the set of components that belong to a partition that a parent decided to delegate). More formally, a mapping needs to be produced, which assigns each $v_k \in V_s$ to a $n_q \in N$ in a way such that the network node n_q is capable of satisfying the requirements and constraints of application node v_k and the edge (v_j,v_k) is mapped to the highest utility link considering all children available at that stage for delegation. The utility of an edge (v_j,v_k) is represented as $U(v_j,v_k)$, and returns the utility achieved due to the mapping of the edge (v_j,v_k) on certain network link. More specifically, the utility of an edge (v_j,v_k), while mapped to the network link (n_p,n_q), where n_p represents the parent in the tree-shaped network where v_j is already mapped and n_q represents a potential child for delegating application component v_k, is calculated by using the following utility function:

$$U(v_j,v_k) = \frac{d(n_q)}{f_1(w_g(v_k) \times w_t(n_q)) + f_2(w_g(v_j,v_k) \times w_t(n_p,n_q))}$$

where $d(n_q)$ represents the number of children of the node n_q, function f_1 models the cost of processing vertex v_k in node n_q and f_2 models the communication cost resulting from mapping edge (v_j,v_k) to link (n_p,n_q).

The utility model in the above scenario is the "highest-degree child with the fastest computation capability and fastest communication link". To ensure that the partitions with the largest number of application components are delegated to the highest degree child, candidate partitions are sorted according to their sizes and then deployed according to that order. In the case of simultaneous scheduling of multiple applications with different priorities, the system needs to guarantee that higher priority applications execute before applications with lower priority. To achieve this, applications are ordered according to their priorities and then mapped following that order. The overall utility of an application graph G with priority p is then calculated as:

$$U(G) = p \times \sum_{(v_j,v_k) \in E} U(v_j,v_k)$$

Therefore, at the level of an individual application the problem of self-configuration becomes the problem of finding highest utility mapping between edges E in the application graph and the Links L of the network graph.

2.5. Self-Optimization

After initial placement, the environment may change and as a result the utility may drop. Thus, it is necessary to monitor the utility and trigger reconfiguration as required. Reconfiguration is triggered in response to a variety of events such as changes in network delays and in bandwidths, changes in available processing capability, etc. Some user specific events may also trigger reconfiguration such as the arrival of a higher priority job, etc. In our design, reconfiguration is performed only within a subtree and therefore is expected to be a less expensive process because of the way the underlying network is modeled. Each parent node periodically measures the workload at each child and its bandwidth to the child and consequently changes the computational and communication weights of that child. By incorporating this monitored information into the utility function, the parent observes the change in utility due to the changes in the network and the compute nodes. As a result reconfiguration is initiated autonomously. Reconfiguration is costly and disruptive, therefore, it is not feasible to initiate reconfiguration unless it is productive. We intend to trigger reconfiguration whenever the utility drops more than a certain threshold (user specified or system generated by comparing the utility during initial deployment).

3. Experimental Evaluation

We evaluated the performance of the self-managed deployment using a simulation study. Our experiments

were performed in a dual, quad-core Xeon processor with 16GB of RAM.

3.1 Simulation Setup

We used GT-ITM internetwork topology generator [14] to generate a sample large-scale, heterogeneous computing environment for evaluating our self-deployment algorithm. We chose their Transit-Stub model that correlates well with the structure of the Internet, including hierarchy and locality. Table 1 lists the relevant parameters of the network topology used in this study. To generate traffic that simulates real world workload and bandwidth consumption in a shared environment, we used the traffic generator available in the ns-2 simulation package [15]. The traffic generator script cbrgen.tcl was used to create 1000 CBR traffic connections between network nodes. The simulation was then carried out for 2000 seconds, we measured link delays (the amount of time required for a packet to traverse a link considering both bandwidth and propagation delay) between the directly connected nodes in the presence of the random traffic over a 10 second period. Based on these snapshots, we then determined the communication weights of the network links in the presence of dynamic traffic.

We ran our tree construction algorithm to create a tree overlay on top of the abovementioned network topology with the application originating machine at the root node. To create the children list, at first we went through all network links and make a list for each node $n \in N$, that has direct connections with n. Our tree construction algorithm then finalized the children list for each network node n, starting from the root node, ensuring that adding a node to n's children list did not create a cycle.

3.2 Experiments and Results

We designed experiments that compared the utility and cost of a deployed application graph using optimal schemes based on the original network topology and global knowledge as opposed to our autonomic approach that uses the tree network and decentralized deployment decisions based on minimal amount of locally available knowledge. In the optimal scheme, the assumption is that a central node monitors every computational and communication resources in the system and, based on this global knowledge makes optimal deployment decision.

Table 1: Network model parameters

The number of Transit Nodes	4
The number of stub nodes/transit node	32
Number of total network nodes	132
Number of total network links	1986
Stub-stub bandwidth	100 Mbps
Transit-transit and transit-stub bandwidth	500 Mbps
Node's processing weight (range)	[20…80]

However, in this approach the central node becomes a bottleneck with a large number of communications arising from constantly monitoring all the resources in the system. Even if it is possible to gather up-to-date information about all the resources at a central node, finding optimal deployment means enumerating every possible mapping of the application components to the network resources and selecting the one that produces optimal results. It also grows exponentially with the number of nodes in the network and the number of vertices in the application graph.

Because of its exponential growth, the above mentioned optimal scheme becomes intractable even for applications with a few components in it. So we developed another semi-optimal scheme that assumed global knowledge but instead of trying every possible mapping it used a greedy approach to limit the number of cases to evaluate. For both schemes, we applied Dijkstra's All Pair Shortest Path algorithm at the central node to calculate the communication weights between every pair of network nodes. We also assumed one-to-one mapping of the graph vertices to the network nodes in all three cases.

The results are presented in Figures 3 and Table 2. Figure 3 compares the utility achieved by all three approaches in case of 4-, 6- and 8-node application graphs and Table 2 reveals the cost associated with them. We do not have any data beyond 8-node in this comparison, as after that, the optimal approach becomes too costly to measure. The results show that the utility achieved by our autonomic approach is on average 30% lower than that of the optimal approach. However, the cost associated with finding the optimal mapping is huge and completely supersedes the benefits of obtaining additional utility by this approach. Figure 3 also illustrates that, in some cases the semi-optimal approach produces lower utility than the autonomic approach. The reason for that is that the greedy heuristics applied in the case of semi-optimal deployment does not yield a global optimum. More specifically, it takes a greedy approach to select the best node at each step, considering all the nodes in the network, and there is a possibility that the best utility node found for delegation at a certain stage may already have been delegated in some former stage.

To evaluate the scalability of our approach, we observed the time taken by our approach to calculate the initial deployment for an increasing number of application vertices and compared them with the time needed by the semi-optimal approach. The results are presented in Figure 4. On average, compared to semi-optimal approach, we observe 90% reduction in the initial deployment calculation time in the autonomic approach. As the application size increases, the cost incurred by our approach is minimal therefore the approach is well suited for larger applications.

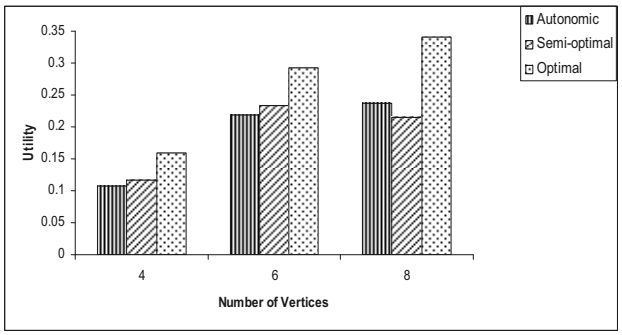


Figure 3. Utility Comparison

Table 2. Execution Time Comparison

# of vertices	Optimal	Semi-optimal	Autonomic
4	8712 µs	15 µs	2 µs
6	15.68 sec	25 µs	3 µs
8	1 hour and 39 minute	34 µs	3.5 µs

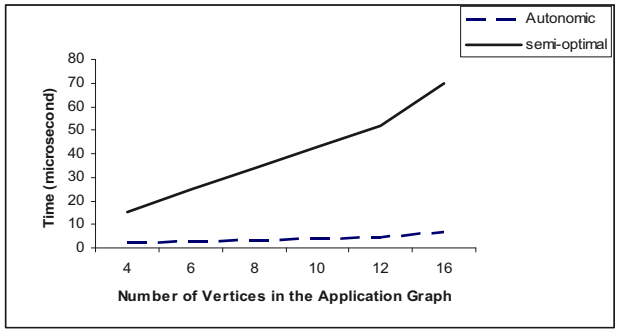


Figure 4. Scalability

4. Conclusion

In this paper, we have developed techniques that enable scalable and efficient deployment of user applications in a highly dynamic and large-scale distributed environment. The approach is to construct an application model, represented as a graph of application components and their interactions and then deploy that graph across the underlying distributed resources organized as a utility-aware tree. A suitable utility function is derived that controls both initial deployment and reconfiguration ensuring that system's overall utility is maximized while certain policies and constraints are satisfied. The main goal of our experimental study was to analyze the tradeoff between optimality and the execution time of our autonomic deployment. The results of our experiments show that considering the tradeoff between the optimal utility and the cost, our deployment algorithm achieves a decent utility with an enormous reduction in the optimization time. Also as the application size increases the cost incurred by our autonomic approach is minimal. Our approach for self-configuration is therefore scalable, robust and more suitable for larger networks and applications. In future, we like to conduct experiments to evaluate our self-optimization approach that dynamically reconfigure the application graph based on the changes in the network.

References

[1] IBM Research. *Autonomic Computing.* http://www.research.ibm.com/autonomic.

[2] W.E. Walsh, G. Tesauro, J.O. Kephart, R. Das, "Utility functions in autonomic systems", *1st International Conference on Autonomic Computing (ICAC),* 2004.

[3] M. Parashar, H. Liu, Z. Li, V. Matossian, C. Schmidt, G. Zhang and S. Hariri, "AutoMate: Enabling Autonomic Grid Applications", *Cluster Computing: The Journal of Networks, Software Tools, and Applications*, Special Issue on Autonomic Computing, Kluwer Academic Publishers, Vol. 9, No. 1, 2006.

[4] X. Dong, S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri, and S. Rao, "AUTONOMIA: An Autonomic Computing Environment", *Proc. of the 2003 IEEE International Performance, Computing, and Communication Conference*, 2003.

[5] D. M. Chess, A. Segal, I. Whalley and S. R. White, "Unity: Experiences with a Prototype Autonomic Computing System", *1st International. Conference. on Autonomic Computing (ICAC)*, 2004.

[6] R. V. Renesse , K. P. Birman , W. Vogels. "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining", *ACM Transactions on Computer Systems*, Vol.21 No.2, 2003.

[7] K. Schwan et al. "Autoflow: Autonomic information flows for critical information systems", *Autonomic Computing: Concepts, Infrastructure, and Applications*, CRC Press, 2006.

[8] D. Deb, M.M. Fuad, M.J. Oudshoorn, "Towards Autonomic Distribution of Existing Object Oriented Programs", *International Conference on Autonomic and Autonomous Systems (ICAS)*, 2006.

[9] O. Beaumont, A. Legrand, Y. Robert, L. Carter, J. Ferrante, "Bandwidth-Centric Allocation of Independent Tasks on Heterogeneous Platforms", *International Parallel and Distributed Processing Symposium (IPDPS)*, 2002.

[10] B. Kreaseck, L. Carter, H. Casanova, and J. Ferrante, "Autonomous protocols for bandwidth-centric scheduling of independent-task applications", *International Parallel and Distributed Processing Symposium (IPDPS)*, 2003.

[11] M. Ripeanu; I. Foster and A. Iamnitchi, "Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design", *IEEE Internet Computing*, Vol. 6, No. 1, 2002.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network", *In Proceedings of ACM SIGCOMM* 2001.

[13] G. Karypis and V. Kumar, "Multilevel k-way Partitioning Scheme for Irregular Graphs", *Journal of Parallel and Distributed Computing,* vol. 48, 1998, pp. 86-129.

[14] GT-ITM: Georgia Tech Internetwork Topology Models. http://www.cc.gatech.edu/projects/gtitm.

[15] The Network Simulator ns-2. http://www.isi.edu/nsnam/ns.

Design of a Fault-Tolerant Job-Flow Manager for Grid Environments Using Standard Technologies, Job-Flow Patterns, and a Transparent Proxy

Gargi Dasgupta[1], Onyeka Ezenwoye[2], Liana Fong[3], Selim Kalayci[4],
S. Masoud Sadjadi[4], and Balaji Viswanathan[1]
[1] IBM India Research Lab, New Delhi, India, *{gdasgupt, bviswana}@in.ibm.com*
[2] South Dakota State University, Brookings, SD, USA, *onyeka.ezenwoye@sdstate.edu*
[3] IBM Watson Research Center, Hawthorne, NY, USA, *llfong@us.ibm.com*
[4] Florida International University (FIU), Miami, FL, USA, *{skala001, sadjadi}@cs.fiu.edu*

Abstract

The execution of job flow applications is a reality today in academic and industrial domains. Current approaches to execution of job flows often follow proprietary solutions on expressing the job flows and do not leverage recurrent job-flow patterns to address faults in Grid computing environments. In this paper, we provide a design solution to development of job-flow managers that uses standard technologies such as BPEL and JSDL to express job flows and employs a two-layer peer-to-peer architecture with interoperable protocols for cross-domain interactions among job-flow mangers. In addition, we identify a number of recurring job-flow patterns and introduce their corresponding fault-tolerant patterns to address runtime faults and exceptions. Finally, to keep the business logic of job flows separate from their fault-tolerant behavior, we use a transparent *proxy that intercepts job-flow execution at runtime to handle potential faults using a growing knowledge base that contains the most recently identified job-flow patterns and their corresponding fault-tolerant patterns.*

Keywords: Software Design, Job-Flow Patterns, Fault Tolerant, BPEL, JSDL, Grid Computing, Peer-to-Peer.

1. Introduction

In Grid and Cluster computing environments, unlike traditional batch environments, individual jobs are typically part of higher-level functional units, generally known as job flows, which are represented by directed graphs. Today, numerous complex academic and commercial high-performance computing applications are being developed as job flows that are composed of several lower-function jobs. Due to the typical long running nature of these jobs, the support for fault tolerance and recovery strategy is especially important.

Very often failure of a job within a flow cannot be treated in isolation and recovery actions may need to be applied to preceding and dependent jobs as well. Thus specifying flow-level recovery mechanisms become important in such scenarios. A prevalent way to handle flow-level compensation is to include failure management logic at modeling time. Wei et al. [1] investigate how to incorporate fault handling and recovery strategy for long running jobs at development time. However, their work requires modification of the original flow to incorporate additional fault-handling logic. The approach also assumes pre-knowledge of all different failure scenarios that can arise. An alternate approach is to handle these job failures at runtime, without explicit changes to job flow process logic. The TRAP/BPEL [2] framework employs this approach for stateless Web service orchestration. In TRAP/BPEL, an intermediate proxy traps calls from the flow engine, and on behalf of it, deploys runtime failure handling. The advantage of this technique is that no direct (or manual) changes need to be made to the flow at development time.

We leverage this approach to enable runtime job failure handling in Grid environments, with dynamic selection of recovery policies. A big challenge in defining recovery policies for Grid jobs is that different jobs may fail at different stages of execution and may require different type of recovery actions. In addition, these long-running jobs often have non-transactional behavior and may require elaborate cleanup phases on account of failure. This is different from the stateless Web service model, where service invocations are of request/response types and recovery plans can mostly be limited to retries of the Web service invocation. Currently, recovery mechanism for long-running jobs requires a high degree of domain expertise. In our work, we explore identification of common, recurrent job flow patterns, and some common fault-tolerance patterns that could be applied to them.

In this paper, we provide a design solution to development of job-flow managers that uses standard technologies to express job flows and employs interoperable protocols for cross-domain interactions among job-flow mangers (Section 2). We enumerate a number of recurring job-flow patterns (Section 3) and

introduce their corresponding fault-tolerant patterns (Section 4) to address runtime faults and exceptions. To promote separation of concerns, we use a transparent proxy that intercepts job-flow execution at runtime to handle potential faults using a growing knowledge base that contains the most recently identified job-flow patterns and their corresponding fault-tolerant patterns (Section 5). Finally, we compare our work to a number of related works (Section 6), provide a short summary and a list of future work (Section 7).

2. Design Using Standard Technologies

It is of primary importance that the design of job-flow managers follows standard and interoperable technologies, such that both the academic and industrial Grid communities benefit from its flexible and open distributed architecture. As part of the Latin American Grid [3], we have developed a two-level distributed architecture that is comprised of two main middleware components: the *job flow manager*, responsible for maintaining concurrency and sequencing among jobs in the flow, and the *meta-scheduler*, responsible for resource selection and job execution control. In the rest of this paper, we will focus only on the design of the job flow manager. Details about the meta-scheduler can be found in [4].

Figure 1 illustrates two resource domains, namely, FIU and IBM and each are managed by their representative job-flow manager along with a meta-scheduler. The assumption is that within each domain, an application or a Web-based portal sends a job flow to the job-flow manager of its respective domain to be executed. The job-flow manager on its turn submits individual jobs to the meta-scheduler on its respective domain; or it sends partial workflows (sub-flows) to a peer job-flow manager in another domain. .

Peering relationships between job-flow managers and between meta-schedulers is established through a set of protocols that exchange dynamic resource capacity and capability information. This enables them to route sub-flows for remote execution at partner domains. The current protocol includes three phases: connection establishment, job-flow submission, and disconnection.

To express the job flows themselves, we chose the Business Process Execution Language (BPEL or WS-BPEL) [5], which has emerged as the standard workflow language for orchestrating service-based applications. Several production-level software from Oracle, Sun and IBM provide core WS-BPEL engines. These engines are virtual machines that interpret and execute WS-BPEL grammar. The grammar models the business logic of the workflow as a directed-graph, where the nodes represent tasks and the edges represent inter-task dependencies, data flow or flow control.

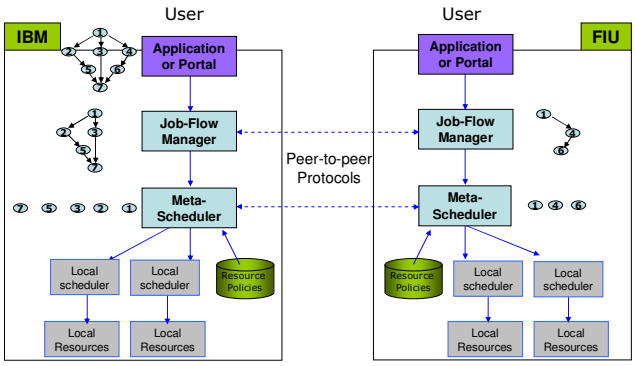


Figure 1: A distributed architecture for flow manager and meta-scheduler spanning multiple domains.

Currently, the BPEL specification does not contain the necessary semantics or support for defining long-running jobs. Grid jobs require the richness and flexibility for specifying varied resource requirements and system environments. The Open Grid Forum job scheduling working group recommends the use of Job Submission Definition Language (JSDL) [6], for capturing a job's resource and environment requirements as well as data dependencies. Ideally, we would like to use uniform modeling and processing semantics at the flow manager and at the meta-scheduler. However, in absence of such unified modeling support, we explore using WS-BPEL and JSDL to provide the combined modeling semantics for job flow. This way individual flow tasks are represented as JSDL jobs, woven together using a WS-BPEL workflow. This provides us with the necessary environment based on standardized technologies to explore the coordination of the flow managers and meta-scheduler for fault-handling purposes.

3. Job Flow Patterns

Figure 2 illustrates a basic set of workflow patterns [7] that are supported by BPEL. In the sequence pattern (Figure 2(i)), an activity in a process is enabled after the completion of another activity in the same process. Parallelism (Figure 2(ii)) allows activities to be executed simultaneously. Loops (Figure 2(iii)) allow for one or more activities to be executed repeatedly. In the choice pattern (Figure 2(iv)), a number of branches are chosen and executed as parallel threads. Based on these basic patterns, more sophisticated constructs can be built [6].

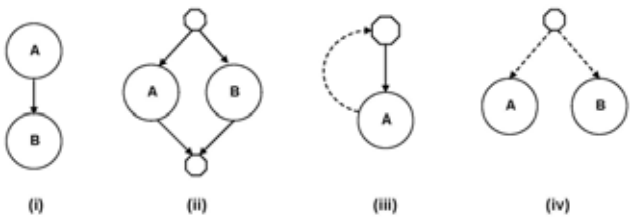


Figure 2: Basic workflow patterns supported by BPEL: (i) Sequence, (ii) Parallelism, (iii) Loop, and (iv) Choice.

In the rest of this section, we present some prevalent patterns arising in job flows. These patterns are stored in the flow patterns repository at the proxy and matched at runtime. Based on the underlying functions, they are categorized into the following:

A. *Job Submission and Monitoring*

A job submission by the flow manager involves invoking the corresponding meta-scheduler interfaces to perform the functions of submission of the job to the resource management layer, and monitoring for any state changes. The different submission patterns observed in job flows include

1. Synchronous job submission: A job flow submits job and waits for completion. In this case the submission call does not return until job completes.
2. Asynchronous submission with polling: A job submission call returns immediately with a job ID. Using the job ID., the job flow polls for the job status.
3. Asynchronous submission with notification: A job flow submits job and gets a job ID. The job flow registers for notification by providing a callback (notification) EPR (End-Point-Reference). The job flow waits for a notification message, before proceeding to the next activity.
4. Asynchronous Fire and Forget: A job flow submits jobs and *does not* wait for job ID or job status (e.g., a batch submission or a cleanup activity). In case of failures, job IDs and job status are sent to the admin or logged instead of invoking job flow. The difference between this and above is:
   - Job flow does not wait for completion of job and thus might complete before such job(s) completes
   - Job failure or success does not affect the job flow business logic

B. *Data Staging*

Many Grid jobs require input data, and in the absence of a shared file system, these datasets need to be staged in at the site of execution. Usually the data staging needs to be completed before the job can begin execution. In case of job-flows, the data requirement could be an input to the system or produced by the execution of a preceding job. In the latter case, a data-dependency is created in the flow between the producer and the consumer jobs of the data. Thus a typical data staging pattern in job flows comprises of staging in data from either producer jobs or from defined inputs, followed by a job submission pattern. There maybe several such data-staging activities, which could occur sequentially or in parallel. Once the data staging of all dependencies are satisfied, a job can be submitted for execution.

C. *Job Execution*

Job execution completion status is captured in the job state and in the job state transitions. Some job execution failures are best handled by looking inside the job definition. For example, if a job failed at 'Data Stage In' state and status message gives which file and its reason it failed to be staged-in (e.g., source not available *or* no space on target), a possible failure handling might involve locating a redundant copy of the file or reserving/freeing space on target resource/filesystem before retrying the job. We generalize this as a job flow pattern where the job execution state helps identify failures and the JSDL job description is used for handling such failures.

4. Fault Tolerant Patterns

In this section, we introduce a classification for exception handling in the job-flows based on patterns introduced in the previous section. The patterns constitute abstract reusable concepts that can be configured for a range of situations. By identifying these patterns, a domain expert can develop a program generator that captures such reusable patterns and can specify which reusable patterns are to be used [8]. The use of a generator in this case would facilitate separation of concerns, that is, the separate addition of fault tolerant concerns to the job-flow. Selected fault-tolerance patterns are then associated with behavioral policies which define the actions to be taken for a failed monitored task. Below, we briefly describe each of these patterns.

Figure 3 shows a state transition diagram that models the patterns identified in Section 3. Explicit data staging activities may precede a job submission. Failure in any one or more of these staging activities entails a transition to the *Failed* state. A successful job submission assumes that the job is ready to be executed. Thus, this state is followed by either polling for job status or waits on job status notifications. On arrival of a job completion notification or change in job state information from the polled job status information, transition is made to the completed stage. At any of the submission or execution stages, a failure would cause a transition to the failed state. In the next few paragraphs, we describe how fault-tolerance patterns can be applied to offer recovery from failures at any of these stages.

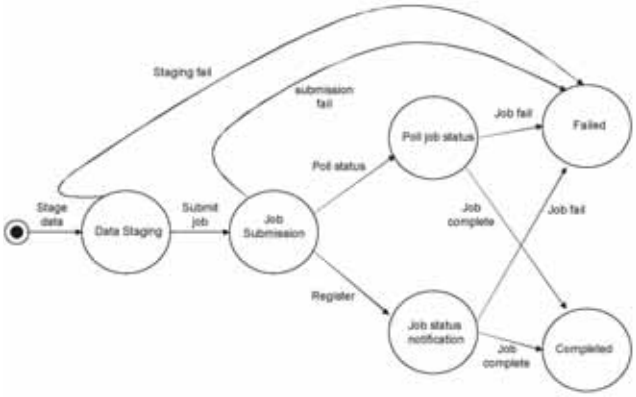


Figure 3: Normal job-flow patterns.

Re-stage data: Data is re-staged upon the occurrence of an exception either at the *data staging* state or during job execution. A job execution failure may explicitly require

the data to be re-staged at the target. Data restaging can be done (a) between the same source and target endpoints of the original staging operation using the same parameters; or (b) by changing the parameters (e.g., data transport protocol, buffer size, timers, etc.) of the transfer; or (c) by specifying a different source in case of multiple copies of the data is present; or (d) by specifying a different target resource when the dependent job is being executed at a new site. Figure 4 illustrates the Re-stage data pattern.

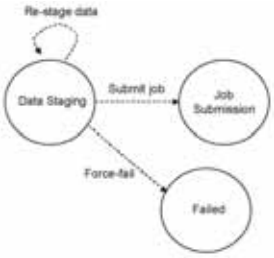


Figure 4: Re-stage data pattern**.**

Re-submit job: A job is re-submitted for execution upon the occurrence of an exception during job submission or execution. Jobs may be submitted to the same or a different domain and may require modifications in job specifications and resource requirements. Submission failures that arise from unavailability of the meta-scheduler can be recovered by submitting to a new domain meta-scheduler. Execution errors require more detailed analysis of job state, status and exit codes and a fair amount of domain expertise for their fault-handling. For example, a domain expert can realize from experience that a job fails due to lack of disk space, and can update the job definition to reflect to request additional disk space. The failed job can be re-submitted with this new requirement. .Figure 5 illustrates this re-submit job pattern. The possible states to transit from here are the *Data Staging*, *Poll Job Status, Job Status Notification* and *Failed* states. .

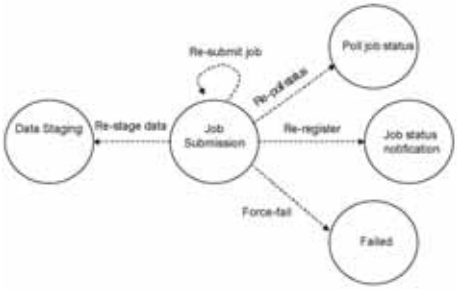


Figure 5: Re-submit job pattern.

Re-poll status: Polling for job status is resumed upon job re-submission. The proxy in this case, uses its co-relation capability to transparently re-poll for the new job re-submission. This involves translating and modifying the original polling messages from the flow to map to the re-polling of the newly re-submitted job. Figure 6 illustrates the re-poll status pattern. The possible states to transit from here are the *Data Staging, Job Submission, Completed,* and *Failed* states.

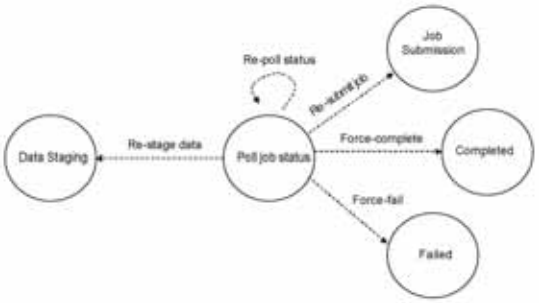


Figure 6: Re-poll status pattern.

Re-register pattern: Proxy registers for callback job status notification after job re-submission. As in case of the Re-Poll status pattern, this re-registration is transparent to the job-flow. Figure 7 illustrates the re-register for notifications pattern. The possible states to transit from here are the *Data Staging, Job Submission, Completed,* and *Failed* states.

Figure 7: Re-register for notifications pattern.

Force-fail pattern: Upon job failure, no further progress is possible and its state is changed to failed [9].

Force-complete pattern: Upon successful job completion, its state is changed to *Completed*. All subsequent activities may now be triggered [9].

5. Fault-Handling Using a Transparent Proxy

As illustrated in the left side of Figure 8, first, the workflow is passed through a Flow Adapter that adapts the BPEL workflow by adding fault-tolerance concerns for specific tasks. The adaptation incorporates some generic interceptors at sensitive join-points in the original BPEL workflow. These join-points are certain points in the execution path of the program at which adaptive code can be introduced at run time. The most appropriate place to insert interception hooks in a BPEL workflow is at the interaction join-points (i.e., the *invoke* instructions). The inserted code is in the form of standard BPEL constructs to ensure the portability of the modified process. This adaptation permits for the BPEL workflow behavior to be modified at runtime [2].

Next, the BPEL based flow manager (FM) executes the adapted workflow. Its main responsibility includes

submission of jobs to the meta-scheduler (MS) and monitoring their progress. Additionally, the notification interface can be used for sending back job state change notifications to the flow manager. Based on resource information at the meta-scheduler, it can decide to execute a job or sub-flow locally or dispatch it to the remote domain for execution. When a sub-flow is dispatched, its execution is handled by the flow manager of the target domain. Jobs dispatched from the flow manager to the meta-scheduler can fail due to several reasons. We broadly classify job failures at the meta-scheduler into the following categories:

1. Job submission failure: In this case, job submission from the flow manager to the meta-scheduler fails for one of several reasons. For instance, the network connection is down, the meta-scheduler is not operational, the meta-scheduler is operational but not accepting new submissions, etc.

2. Job execution failure: In this case the meta-scheduler queues the job for submission, but the job fails during execution for reasons that may include resource unavailability, data unavailability, incorrect input specification, internal job exceptions, output data staging, and exceptions during cleanup.

As illustrated in the right side of Figure 8, for runtime failure management at the level of individual jobs, we use a transparent proxy, introduced in TRAP/BPEL [2]. In this case, the proxy sits between the flow manager and the meta-scheduler, and intercepts calls in both directions. For all monitored invocations, the meta-scheduler interface calls are replaced with calls to the proxy interface. However, the proxy is transparent to the flow manager and to the meta-scheduler; therefore, it imposes no changes in either component. The proxy exposes a generic interface to the flow manager which accepts messages containing original invocation parameters, marshaled by the adapter.

A transparent proxy comprises three distinct components: (1) A monitoring component that monitors each adapted invocation; (2) A message correlator component, which correlates individual messages flowing through the proxy to construct conversational state; and (3) A recovery component that kicks in when failure is detected for any adapted component.

An extensible repository of job-flow as well as fault-tolerant patterns is maintained at the proxy. Job flow patterns comprise of common artifacts that are prevalent in job flows represented using the combination of a flow language and a job definition language (e.g., a job submission activity is typically followed by a monitor job state activity). The proxy by virtue of maintaining conversational state for each job is well equipped to detect and handle failures. Fault-tolerant patterns comprise common reusable recovery actions that can be specified for job flow failures. The mapping between job-flow patterns and fault-tolerant patterns can be manually defined at modeling time by the application developer or using pre-defined rule trees. Depending on the rules specified in the tree, a choice can be made on which fault-tolerance pattern to use depending on the job flow pattern. Rules could also be based upon runtime information and domain knowledge.

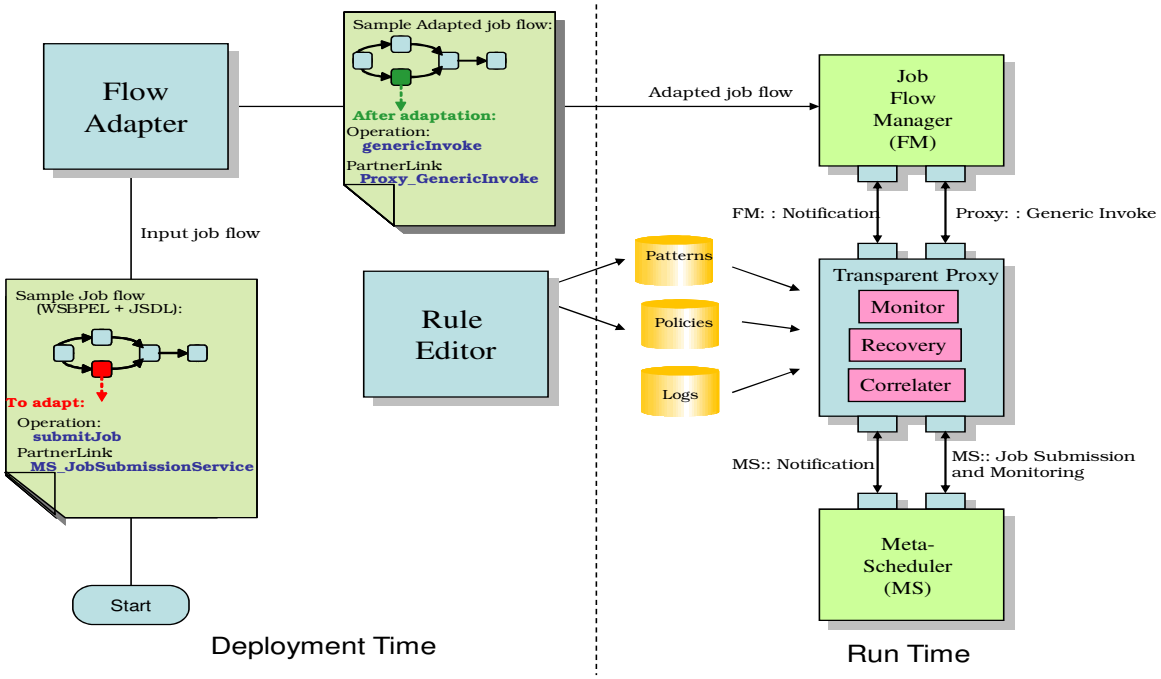


Figure 8: The fault-tolerant architecture using a transparent proxy

6. Related Work

BPELJ [10] is an extended version of BPEL. Java snippets can be included in BPEL processes for business logic or fault-tolerance concerns. This approach has portability problem, since it needs a specific BPEL engine. AdaptiveBPEL [11] follows an aspect-oriented approach for dynamically adapting a Web service to provide both functional and QoS customization. Adaptation process is policy-driven similar to ours, but this approach also needs a specially built BPEL engine. Pegasus project [12] provides a framework for constructing workflows and mapping these workflows onto Grid resources. Even though Pegasus has advanced capabilities for a better performance of workflow execution, less is provided in fault-tolerance aspect. It provides only remapping of an entire sub-flow in case of a failure whatever the reason may be. The prototype BPEL4JOB [1] also investigate how to incorporate fault handling and recovery strategy in WS-BPEL for long running jobs at modeling time. An alternate approach is to handle these failures at runtime. Authors in [13] study the impact of runtime optimizations made at the scheduler for handling workload surges, while minimizing the reconfiguration overhead.

7. Conclusion and Future Work

In this paper, we presented a design for a fault-tolerant job-flow manager that can handle failures at runtime using standard protocols, job-flow patterns and a transparent proxy. We identified common job-flow patterns and some reusable fault-tolerant patterns that can be used for their recovery. We discussed the processes required at development time for a successful runtime fault-tolerant behavior in job flows. In future work, we plan to evaluate our work using a comprehensive set of failure scenarios, explore automatic generation of mapping between job-flow patterns and fault-tolerant patterns, and study the performance impacts of some of these fault-tolerant patterns.

Acknowledgements

This work was supported in part by IBM, the National Science Foundation (grants OISE-0730065, OCI-0636031, REU-0552555, and HRD-0317692).

References

[1] W. Tan, L. Fong, and N. Bobroff. Bpel4job: a fault-handling design for job flow management. In Proceedings of Fifth International Conference on Service Oriented Computing (ICSOC), 2007

[2] Onyeka Ezenwoye and S. Masoud Sadjadi. TRAP/BPEL: A framework for dynamic adaptation of composite services. In Proceedings of the International Conference on Web Information Systems and Technologies (WEBIST 2007), Barcelona, Spain, March 2007.

[3] Rosa Badia, Gargi Dasgupta, Onyeka Ezenwoye, Liana Fong, Howard Ho, Sawsan Khuri, Yanbin Liu, Steve Luis, Anthony Praino, Jean-Pierre Prost, Ahmed Radwan, Seyed Masoud Sadjadi, Shivkumar Shivaji, Balaji Viswanathan, Patrick Welsh, and Akmal Younis. *High Performance Computing and Grids in Action*, chapter Innovative Grid Technologies Applied to Bioinformatics and Hurricane Mitigation. IOS Press, Amsterdam, 2007.

[4] Norman Bobroff, Liana Fong, Selim Kalayci, Yanbin Liu, Juan Carlos Martinez, Ivan Rodero, S. Masoud Sadjadi, and David Villegas. Enabling interoperability among meta-schedulers. In *Proceedings of 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid-2008)*, Lyon, France, 2008.

[5] Ezenwoye, O., Sadjadi, S.M.: Composing aggregate Web services in BPEL. In Proceedings of The 44th ACM Southeast Conference, Melbourne, Florida (2006).

[6] A. Anjomshoaa, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. *Job Submission Description Language (JSDL) Specification, Version 1.0*. Global Grid Forum, 2005.

[7] Dieter Cybok. A Grid workflow infrastructure: Research articles. Concurrency and Computation: Practice and Experience, 18(10):1243–1254, 2006.

[8] Ian Sommerville. Software Engineering, 8th Edition; Chapter 18: Software Reuse. Addison Wesley, May 2006.

[9] N. Russell, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Exception Handling Patterns in Process-Aware Information Systems. BPM Center Report BPM-06-04 , BPMcenter.org, 2006.

[10] Michael Blow et al, BPELJ: BPEL for Java, A Joint White Paper by BEA and IBM, March 2004.

[11] Erradi, A.; Maheshwari, P.; Padmanabhuni, S. Towards a policy-driven framework for adaptive Web services composition, Next Generation Web Services Practices, 2005.

[12] Ewa Deelman et al. Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems, Scientific Programming Journal, Vol 13(3), 2005, Pages 219-237.

[13] G. Dasgupta, K. Dasgupta and B. Viswanathan. Data-WISE: Efficient management of data-intensive workloads in scheduled Grid environments. To appaer in Proceedings of *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2008.

Supporting Context-Awareness in Web-Based Groupware Development

José Maria N. David[1], Marcos R. S. Borges[2], José A. Pino[3]
[1]*Faculdade Ruy Barbosa, Salvador, BA, Brazil*
josemaria@frb.br
[2]*Federal University of Rio de Janeiro, PO Box 2324, 20001-970, Rio de Janeiro, RJ, Brazil*
mborges@nce.ufrj.br
[3]*University of Chile, Department of Computer Science, Chile*
jpino@dcc.uchile.cl

Abstract

Context in groupware development has been evaluated in some standalone applications in an ad-hoc manner. Although one of the main goals of groupware infrastructures deployment is to provide the necessary flexibility to build groupware applications, most of them do not supply enough mechanisms in order to consider the context in which group participants interact. Consequently, a large amount of not relevant information could be presented to the user. This paper presents a brief discussion about the importance of context elements support in groupware. Some context features already available in a groupware infrastructure are discussed, in order for them to be deployed in a web-based groupware development. As part of this infrastructure, a context-awareness service which considers group context is also described.

Keywords: Awareness, Context, Context-Awareness, groupware, CSCW.

1. Introduction

Groupware infrastructures have been considered as one of the ways to design and develop suitable groupware applications. Toolkits have been developed and used in order to supply the basic components for the development of such applications. However, one of the challenges to this approach is that CSCW literature reports few cases of success and possible flaws in large-scale groupware toolkits use [6]. They do not consider challenges related to significant group awareness information, nor contextual features.

Most toolkits provide support only to collaborative actives and are not flexible enough to address the social aspects related to the interaction of geographically dispersed groups. Usually, they do not consider the need to change awareness elements accordingly. This means that they lack mechanisms capable of considering the dynamic context in which collaborative activities are accomplished. According to Dourish [5] context is not stable along with group interaction, and depends on the circumstances within which it occurs.

For example, consider meeting support systems and their awareness mechanisms, such as, event log, user list, contribution graphic chart, and so on. Although these mechanisms presumed by the designer are important, only coordinators are interested in graphic chart visualization. However, at a latter moment, they could be interested in another chart aimed to focus on different contribution categories (for example, only issues and positions). This means that, previous contribution graphic chart is no longer relevant as activities proceed. Therefore, it is difficult for groupware designers to presume a set of awareness requirements good enough to fulfill group requirements and, consequently, to promote an adequate level of group interaction. At the same time, they need to supply all context information according to each activity without overloading the workspace.

The goal of this paper is to describe a work aimed to provide a context-based awareness service to be coupled in a web-based groupware infrastructure [3]. This service provides some features which are available as an integrated tool framework, supplying conceptual tools and guidelines to account for context-awareness when developing web-based groupware applications. Through this service we hope to provide the necessary support for the complex dynamic task considering: the contextual knowledge of each participant, and its persistence in group memory, as well as its evolution according to each integrated groupware application in different contexts.

Our research interest lies at context issues that emerge along group interactions. Particularly, we are not aimed at discussing context elements which surrounds user physical environment [4].

This paper is organized as follows. Section 2 presents related works both to awareness filtering and context in groupware activities. Section 3 presents an infrastructure aiming to support web-based groupware applications design. In Section 4, the proposed context-awareness model is discussed. In Section 5, implementation issues as well as an example of the possible profiles which could be established through the

context-awareness service are addressed. Conclusions are presented in Section 6.

2. Related Work

Most groupware applications proposed in the literature discuss awareness mechanisms deployment based on event notification to provide the user with information about important cooperative activities [7, 8, 10]. These platforms deal with specific features to support awareness in shared workspace by using pre-established components. They hardly ever present features capable of ensuring that a specific support is generic and flexible enough, so that the user can modify his/her workspace according to his/her activities and interests in the target context.

Groupkit [9] was constructed in order to provide support for synchronous groupware applications. However, unlike our approach, this toolkit provides neither generic components for cooperative applications capable of being reused in the web, nor dynamic support to context-awareness.

In the Atmosphere framework [8], pre-defined context (spheres) allows participants to configure group contexts, as well as to select suitable context while performing an activity. However, as far as we are concerned, Atmosphere does not consider integration issues between collaborative applications, dealing especially with asynchronous groupware use.

3. COPSE-Web Infrastructure

Aiming at supporting the development of web-based groupware application, COPSE-Web was designed. A tool framework provides facilities for groupware development considering integration and interoperability issues. However, its infrastructure is not prepared for a web-based application development support and lacks many important features, which will be required when developing the complex asynchronous applications.

COPSE[1]-Web infrastructure was designed in order to fulfill the requirements of both synchronous and asynchronous cooperative activities, which are accomplished in the web-based environment [3]. Being a comprehensive framework, COPSE-Web provides facilities for the development of fully integrated web-based groupware applications with fundamental services, such as communication, coordination, awareness and group memory.

COPSE-Web awareness components can be launched in the environment or instantiated from its framework in any groupware application. When instantiated in the environment, awareness components can be used by any participant who is logged on, but he/she does not necessarily start any application. To illustrate the point, we can mention the bulletin board and event log components. They gather data related to the interactions, which are accomplished in the environment.

However, these components were not capable to support dynamic context issues in work group. These challenges make us notice a need for an infrastructure capable of offering an architectural support for groupware designers, using adequate mechanisms which could represent relevant information about context in any way. Hence, we need alternative forms to select and represent context according with interaction evolution.

4. The Proposed Context-Awareness Model

Aiming to fulfill awareness requirements that should be provided by groupware, we suggest that a context-based model be observed. We claim that representational challenge related to the adequate set of awareness mechanisms in the shared workspace can be supported. At the same time, we aimed to consider context as a dynamic issue supplying awareness components filters to be established along with group interaction.

Figure 2 illustrates the process where User 1 generates information (I1 and I2) which is represented by several awareness components in some way. For instance, considering a collaborative discussion forum, I1 and I2 could be related to the contributions accomplished by *User 1*. By the selection of an adequate set of awareness components *User 2* and *User 3* could understand and interpret the work situation accordingly. At the same time, both of them could focus their attention on the key workspace area. Otherwise, *User 2* realizes I1 selecting the awareness component "contribution report" (C1), which presents the average value of each contribution categories (i.e., question, position or argument) of *User 1*. Subsequently, *User 3* realizes the same information (I1) with the awareness component "contribution meter" [1], but in a less detailed presentation way (C2).

This action is possible because awareness components are selected according to their role and responsibilities. So, as a coordinator, *User 1* can identify those participants that contributed as expected. For example, if the percentage of questions is high, the coordinator could analyze the awareness profiles generated by each user selection before any motivated actions, eventual conflicts resolution, or before supplying additional contextual information [3].

[1] COPSE – *COllaborative Project Support Environment*

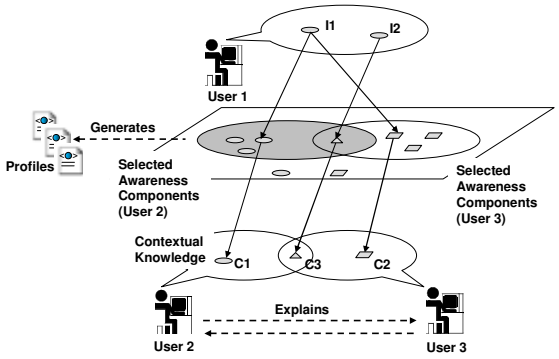


Figure 2. Profiles, awareness and context in collaborative activities.

In addition, both users realize the information by the selection of the same awareness component (C3); for example, the "events report" or the "bulletin board". Consequently, both of them can realize the information I2 with the same focus and point of view (related to C3). As a result, they can interact easily, and can explain the work situation arising from the interaction to each other, because they realize the same context. This means that they invoke part of the contextual knowledge to anticipate some actions and decisions [2]. As a result, they reduce the information overload in the workspace disregarding useless information.

This model can be deployed both on application design and configuration along with group interaction. In other words, filters could be deployed – and combined –, from application to users' direction and vice-versa. From application towards users, filters are usually established in order to fulfill users' requirements and presumed context. On the opposite direction, filters are defined aiming at selecting which information needs to persist on the database. On the users' side, filters are initially deployed according to the group and each needs of the role. In the course of the interaction, they are actively changed according to the context. This may change over time as users become skilled, requiring a different set of information when performing their activities. When combined, these filters (user, group and role) produce profiles (user, group and role) aiming at supporting reciprocal awareness. Using such a dynamics we attempt to support work group context not only as a representational challenge, but rather, as Dourish [5] says, as an "interactional problem".

5. Implementation Issues

In order to provide an infrastructure capable of supplying the necessary functionality as discussed previously, we have designed an architecture aiming to fulfill context requirements. As a part of this infrastructure, a context-based awareness service was designed aiming to provide adequate awareness information related to the tool context as well as the users' and group's contexts. This service was added to the COPSE-Web tool framework, based on the previously discussed model.

Our claim here is that in order to solve the awareness problem that was stated previously, two points should be addressed: (i) the selection; and (ii) the awareness artifacts presentation in workspace. Information presentation is related to the context in which collaborative activities are accomplished. On the one hand, awareness mechanisms could be adequately designed from the knowledge of this context. On the other hand, however, appropriate data visualization for the application and for the participant's role could make each participant aware of the necessary context to accomplish his/her activities. As a result, information overload could also be reduced and, at the same time, modifications in the group interface could reflect preferences, behaviors and context of the group.

Besides the fact that some servers have already coupled to the environment, in COPE-Web architecture we propose the development of an additional server: the *Profiles and Awareness Server*. This server is directly associated to the profile mechanisms which offer generic awareness components for the environment, or specific awareness components for groupware applications. Profile mechanisms module supplies the necessary functionality for the construction, storage and queries of awareness profiles. Through the profile analysis we intend to supply suitable awareness components for an activity to be accomplished. The awareness components are directly related to the group participants' preferences and individual interests (personal profile), to the role carried out by him/her (role profile) or to the goals and expected results for the group (group profile).

Profiles are built as activities are accomplished. They are represented as XML files. If updated frequently, according with filters composition (application, users and role filters), their analyses are sources of information to guide coordination actions.

5.1. Example of Profiles and Context in COPSE-Web Infrastructure

Several groupware applications have been implemented using the COPSE-Web framework. They have been developed under the context-awareness service concept. For example, a pre-meeting support system was developed considering the contextual elements available in the COPSE-Web framework. Developers were capable of reusing components from Awareness package. Among these components we can mention the event log component in which events related to the carried out activities are shown according to the user's interest.

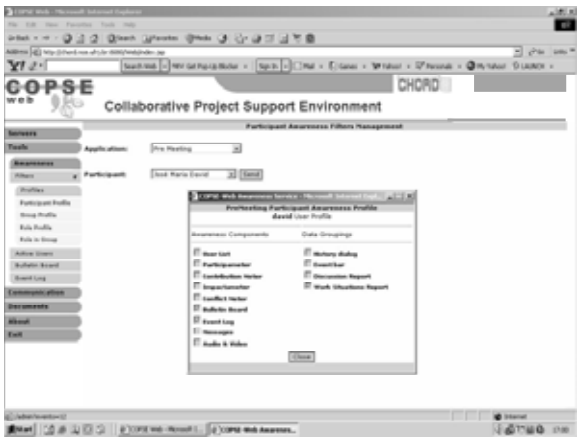


Figure 3 - Example of Awareness Service Interface (participant profile generation)

Other awareness mechanisms were deployed at the design time, for instance; graphical charts representing not only the participation and contribution rates, but the impact of an item. Suppose that only coordinators are interested on information concerned with each of these mechanisms. According to the presented model, they are filtered by every participant. However, if two coordinators (User 2 and User 3, for example) do not filter the bar chart which presents the percent of issues raised by each participant, both of them visualize the same information with the same focus. In addition, through the profile analysis each of them knows that the other knows that information allowing, as a consequence, that their contexts could be aligned.

The profile package aims to extend the context support in COPSE-Web. Part of the profile package is the mechanism of selection of awareness components generating profiles. As Figure 3 illustrates, by selecting one of the four types of profile items (participant, group, role and role in group), each participant can view all the awareness and data grouping components which were filtered by the chosen application, in advance.

6. Conclusions

Context in groupware development has been considered in stand-alone – non-integrated – applications. Using COPSE-Web framework, developers may benefit from some tools, models and guidelines already available, leading them to the context representation. At the same time, we aimed at preserving the inherent dynamic behavior of web-based groupware application providing awareness elements to each associated work group context, namely: individual and role context, as well as group context. This issue has been possible not only using the designed architecture but also using the available selection mechanisms, which filters awareness elements and generates different profiles. Furthermore, the analysis of these profiles supply context information for both the participants and the coordinators.

We have evaluated the awareness service in a meeting context. Case studies have frequently been carried out in pre-meeting systems. The main issue observed is related to the work burden for each participant when analyzing awareness profiles. Coordinators have generally reported the work overload during collaborative activities to decide the most appropriate action. This fact becomes intensive when the number of participants increases. So, we have to evaluate the proposed solution taking into account small groups.

References

[1] M. R. S. Borges, J. A. Pino, "Awareness Mechanisms for Coordination in Asynchronous CSCW". *WITS'99*, Charlotte, N.C., 1999, 69-74.

[2] M. R. S. Borges, P. Brézillon, J. A. Pino, J. Ch. Pomerol, "Dealing with the effects of context mismatch in groupwork". *Decision Support Systems*, 43(4): 1692-1706, 2007.

[3] J. M. N. David, M. R. S. Borges, "Designing collaboration through a web-based groupware infrastructure". *International Journal of Computer Applications in Technology*, v. 19, n. 3/4, 2004, 175-183, Inderscience.

[4] A. K. Dey, G. D. Abowd, D. Salber, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications". *Human-Computer Interaction (HCI) Journal*, 16(2-4), 2001, 97-166.

[5] P. Dourish, "What We Talk About When We Talk About Context". *Personal and Ubiquitous Computing*, 8(1), 2004, 19-30.

[6] S. Greenberg, "Toolkits and interface creativity". *Journal Multimedia Tools and Applications*, 32(2), 2007, 137-234.

[7] W. Prinz, "NESSIE: An Awareness Environment for Cooperative Settings". *Proc. of the 1999 Sixth European Conference on Computer-Supported Cooperative Work (ECSCW)*, Copenhagen, Denmark, Kluwer, Sept., 1999, 391-410.

[8] M. Rittenbruch, "ATMOSPHERE: A Framework for Contextual Awareness". *International Journal of Human-Computer Interaction*, 14(2), 2002, 159-180.

[9] M. Roseman, S. Greenberg, "Building Real Time Groupware with Groupkit, a Groupware Toolkit". *ACM Trans. on Computer-Human Interaction*, 3(1), 1996, 66-106.

[10] M. Sohlenkamp, W. Prinz, L. Fuchs, "POLIAwaC: Design and Evaluation of an Awareness Enhanced Groupware Client". *AI & Society*, Vol. 14, 2000, 31-47.

"Object-Z to Java/OO Perl": A Conversion from Object-Z to Executable Skeletal Code with Dynamically Checkable Design Contracts

Sherri M. Sanders* and Cui Zhang
Department of Computer Science
California State University, Sacramento, CA 95819-6021
sherri.sanders@hp.com* zhangc@ecs.csus.edu

Abstract

Designing reliable, correct, and robust programs is a challenge today. By using formal specifications for requirements and design-by-contract at the program design level, software developers can define systems using unambiguous syntax and semantics. This leads to software that can be rigorously verified and analyzed, ensuring that the reliability, correctness, and robustness of the systems can be significantly improved. This paper presents a tool called "Object-Z to Java/OO Perl", which provides an automated conversion from Object-Z specifications to executable Java and object oriented Practical Extraction and Report Language (OO Perl) skeletal code with dynamically checkable design contracts. This conversion extends Java and OO Perl with the design-by-contract mechanism and bridges the semantic gap between formal specifications in Object-Z and design contracts at the programming language level.

1 Introduction

Software program designers are challenged to produce software that, over time, across platforms and through modifications and upgrades, will remain reusable and reliable. Reliable software touts two major features: correctness and robustness.

Unfortunately, requirement specifications that use natural language can be ambiguous and lead to numerous errors when developing large, complex software programs. By using formal specifications for requirements and design-by-contract [9,10,14] at programming language level, software developers can define systems using unambiguous syntax and semantics, which can be rigorously verified and analyzed, therefore; the correctness and robustness of the systems can be significantly improved. This paper presents a tool called "Object-Z to Java/OO Perl". It utilizes the advantages of a formal framework by extracting the exact specification in Object-Z [15] and generating Java and OO Perl skeletal code. It captures the formally specified requirements and translates them into skeletal code with dynamically checkable design contracts [9,10,14]. Originally, design-by-contract is not a built-in mechanism of either Java or OO Perl.

The tool "Object-Z to Java/OO Perl" is an enhancement to the previous tool Object-Z-to-Java [13], which is limited by its use of only a small subset of Object-Z data-types, i.e., primitive data-types. The development of both tools was inspired by the research of several available works on structural mappings from Object-Z to C++ [3,6,12] to Eiffel and to Java [4], as well as from CSP to Java [1].

The tool presented by this paper incorporates additional Object-Z data structures, which includes sets and set operations. It provides a conversion from Object-Z specifications to executable Java and OO Perl skeletal code with dynamically checkable design contracts. It also improves the Graphical User Interface (GUI) to facilitate usability. Java and OO Perl are chosen for this conversion because of their extensive use in development.

The Object-Z framework is divided into units or blocks for basic type definitions, global constants and/or variable definitions, and schemas. Schemas define the system's state and its operations. Each class schema has a class name, the possible generic parameters, and their properties. The Z schemas are extended to define classes, thereby, providing a clear visual representation of the scope of the definition.

"Object-Z to Java/OO Perl" can extract the design contracts from formal specifications in Object-Z and map them directly to the implementation skeletal code with dynamically checkable design contracts. This approach blends the benefits of formal methods with the strength of design-by-contract and it provides the ability to test and directly relate assertion error messages of the executable code to the formal specification. By providing a connection between formal specification and implementation code, this tool is effective in helping the programmers bridge the semantic gap between distinct formal specification and dynamically executable design contracts at the code level.

Tools supporting formal methods range from "heavy weight" model checkers and theorem provers to "light weight" type checkers/type setters such as ZML document markers (XML for Z) and easy-access browsers for formal specifications [16]. The development of more "light weight" tools, such as "Object-Z to Java/OO Perl" can help gain a broader

* Currently working at Hewlett Packard, 8000 Foothills Blvd, Roseville, CA 95747

user base for the application of formal methods in software development practice.

2 The "Object-Z to Java/OO Perl" System

2.1 Overview of the System

The "Object-Z to Java/OO Perl" system provides the following:

- A direct mapping of design contracts from formal specification level to programming language level, giving developers the opportunity to focus on the system's functional requirements specification
- A graphical representation of the Object-Z specification to aid in defining the system functional requirements
- A mapping from Object-Z to either Java or OO Perl to assist the understanding of the relationships between formal specifications in Object-Z and design contracts in OO programming languages
- An extension to Java and OO Perl with the design-by-contact mechanism to aid in dynamic program analysis

A graphical user interface (GUI) interacts with system developers to capture Object-Z specifications. The GUI can help developers become proficient in the Object-Z notation and provide a level of abstraction between the analysis and design of the contracts and their implementation.

The raw input data for the Object-Z specification collected from the GUI is converted to "tagged" data elements in an XML (eXtensible Markup Language) document. W3C [17] developed XML and the methods for defining the XML-based data models. The XML DTD (Document Type Definition) is the model used in this system. The simple text-based solution provides an easy to learn implementation model, which is made up of a set of fundamental units or building blocks [17] used for data validation.

The "Object-Z to Java/OO Perl" system uses a rule-based approach for mapping the Object-Z data structures to Java and OO Perl. The XML DTD captures the characteristics, such as visibility, type, structure, and name, of each Object-Z element. The "Object-Z to Java/OO Perl" system then determines the mapping structure from Object-Z to executable skeletal code based on the specified characteristics.

"Object-Z to Java/OO Perl" provides a direct mapping of design contracts from the formal specification to the OO programming. As an extension to [13], Table 1 shows the mapping from Object-Z to both Java and OO Perl. As defined by A. Harry [5], the following data types and structures make up the Object-Z domain: 1). User-defined identifiers, 2.) Data types, 3.) Basic types such as predefined types and sets, given types, free types, 4.) Compound types such as sets, bags, and sequences, and 5.) Schemas. The "Object-Z to Java/OO Perl" system handles data-types including sets, sequences, bags, and operations on these data-types.

Mapping of Object-Z to Executable Skeletal Code	
Classes	Functionality
Class Schema Name	Public class "CLASSNAME" - *Java* Package "CLASSNAME" - *Perl*
Class Constants	Final class constant values - *Java* Use constant values - *Perl*
Class Variables	Class variables - *Java / Perl*
Class Invariant	Code for checking the invariant at object initialization and before and after each method invocation - *Java / Perl*
Initial Schema	Class Constructor - *Java / Perl*
Operation Schema 1. *Modifiers* 2. *Pre-Conditions* 3. *Post-Conditions* 4. *Implementation Code*	Class Methods - *Java / Perl* 1. List of variables that does or does not change 2. Code for checking pre-conditions that must be satisfied before the method is executed 3. Code for checking post-conditions that must be guaranteed after the method is executed 4. Implementation code to be provided by programmer
Visibility List	"public, private, protected" methods/variable modifiers - *Java* "our, my" variable moditiers - *Perl*

Table 1: Mapping between Object-Z and Java/OO Perl

2.2 Software System Architecture

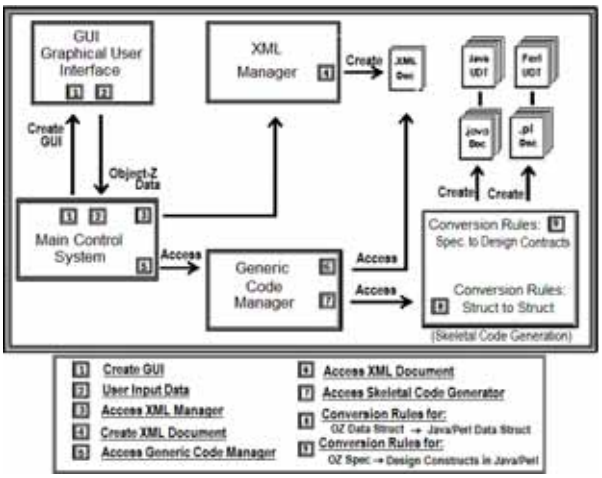


Figure 1: Software System Architecture

The software system architecture for "Object-Z to Java/OO Perl" is illustrated in Figure 1. The functionality of the components is listed below.

- *Main Control System (MCS)* – The MCS is the primary control system that creates and presents a series of successive GUI panels that capture the input for the Object-Z specification. The input passed to the MCS is piped to the XML manager. When the user has completed entering data for the Object-Z class schema, a well-formed XML document, ("CLASSNAME".xml) is created. That XML document is used by the "Skeletal Code Generator".

- *Graphical User Interface (GUI)* – The GUI is composed of a series of successive Java Swing Applets with a menu structure. The initial applet displays buttons that provide the user the option of creating a new Object-Z schema or generating a skeletal class from a previously created XML document. It also presents a menu where the user can create a new Object-Z schema and traverse that path, exit to the main page or exit the system.
- *XML Manager* – The XML manager is responsible for creating the XML document for the system. Each time it is called by the MCS, new raw data is passed to it and the manager then recreates the XML document with the existing data and the new input. Any changes to previously entered data will be queued as new data. This subsystem is made up of an XML *DTD file*, the class.dtd file, which defines the XML tag structure by listing all permissible elements. This document is essential to ensuring that data is validated and the "CLASSNAME".XML document is well formed.
- *Generic Code Manager* – This generic code manager creates a new SAX parser object, which accesses and processes the "CLASSNAME".XML file. The data is then tokenized by the SAX parser and passed on to the Skeletal Code Generator.
- *Skeletal Code Generator* – The skeletal code generator uses a set of rules to convert the Object-Z data types to either Java or OO Perl data types. It then converts the Object-Z specification to design constructs in either Java or OO Perl (language determined by user).

2.3 Conversion Rules

The conversion from Object-Z data structures to Java and OO Perl data structures is accomplished by a set of conversion rules. The rules originate from the tagged elements of the XML document. These rules are individually based on the target programming language. The conversion of basic types, numeric operations and Boolean operations from Object-Z to Java, requires only the use of Java's predefined, primitive types as well as Java's arithmetic and Boolean operations.

Perl is very similar, with regard to converting basic types; however, it is unnecessary to declare a variable type or size. Although specific symbols are used to characterize variables, they do not need to be typed or type cast. $VAR represents a scalar or variable, @VAR represents a list or array, and %VAR represents an associative array or hash.

One advantage of using Perl is the use of the hash abstract data structure. Perl's hash data structure is composed of a collection of key/value pairs, where each key is associated with exactly one value. This allows a one to one mapping of the Object-Z set operations "dom" and "ran", where the Perl hash key maps to the return value of the "dom" function and the Perl hash value maps to the return value of the "ran" function. Java also provides a one to one mapping when using an ArrayList of lists, similar to a multidimensional array.

Mapping compound structures to Java requires a different process. In the "Object-Z to Java/OO Perl" system, state and formal elements can take any basic or compound structure. If an element type is characterized as anything other than a variable, it is converted to an ArrayList in Java and either an array or a hash in Perl. The ArrayList is used instead of the array data structure because the ArrayList can contain an element of any type without being predefined.

Table 2 shows two mapping examples for both basic and compound types from Object-Z to Java and OO Perl.

Mapping Example from Object-Z to Java and Perl			
XML	Object-Z Schema	Java	Perl
Visibility = No *Type = Integer (Z)* *Structure = Variable* *Name = maxSize*	maxSize : Integer (Z)	private int maxSize;	my $maxSize;
Visibility = Yes *Type = String* *Structure = Power Set (P)* *Name = lib*	lib : Pset String	ArrayList lib;	our %lib;

Table 2: Mapping Example from Object-Z to Java and OO Perl

There is a mapping for set operations from Object-Z to Java and OO Perl. Table 3 illustrates the mapping from Object-Z for representative set operations to Java and OO Perl. In this table, "enrolled" is a set of User-Defined Type (UDT) STUDENT, "s?" is a method input parameter of UDT STUDENT, "#enrolled" is the number of elements in the set "enrolled", "maxSize" is a constant with some defined integer value, and "enrolled" along with "enrolled_New′" (enrolled prime) represent the "enrolled" set with a change in state.

In the Pre-condition case in Table 3, the ArrayList class is used for implementing an Object-Z set structure in Java. This allows an element of any type to be inserted or removed from the list. "Contains", which is a method provided by ArrayList, returns true if the list contains the specified element. When mapping a list structure, to either Java or OO Perl, an Assertions class is created for processing a subset of Object-Z compound set operators. The Assertions class implements a set of Boolean methods, two of which are, "memberOf" and "bagUnion".

These Assertions class methods are called from within the class invariant, pre-condition or post-condition.

Mapping from Object-Z Set Operations to Java and Perl			
XML	Object-Z Schema	Java	Perl
Pre-condition			
PreCondVar1 = s?			
PreCondRelOp = "∈"	Pre: s? ∈ enrolled	enrolled.contains(s);	memberOf (s, enrolled)
PreCondVar2 = enrolled			
Class Invariant			
ClassInvariants1 = maxSize			
ClassInvariantsRelOp = "<="	#enrolled <= maxSize	enrolled_Size <= maxSize	$#enrolled_Size <= maxSize
ClassInvariants2 = #enrolled			
Post-condition			
PostCondVar1 = enrolled ∪ {s?}			
PostCondRelOp = "="	enrolled' = enrolled ∪ {s?}	bagUnion (enrolled_New, s, enrolled)	bagUnion (*enrolled_New, $s, *enrolled)
PostCondVar2 = enrolled'			

Table 3: Mapping from Object-Z Set Operations to Java and OO Perl

When mapping a set structure to Java, a number of additional elements are defined, such as "ArrayName", "ArrayName_New, "ArrayName"_Size, "ArrayName"_New_Size. The Perl language provides the same functionality by referencing the array and array size as, %"ArrayName", %"ArrayName_New, $#"ArrayName"_Size, $#"ArrayName"_New_Size". In the Class Invariant case in Table 3, for mappings to both Java and OO Perl skeletal code, the number of elements is compared to maxSize and returns true as long as the value is less than or equal to maxSize.

In the final Post-condition case in Table 3, the method "bagUnion", requires three input elements, the old list, the new list, and the element that was added to the old list. It performs a comparison between, the "enrolled_New´" list and the union of the old "enrolled" list with the input element "s".

2.4 Implementation

The programming language used to develop the "Object-Z to Java/OO Perl" system tool was Java, version 1.4.2_14. BlueJ 2.2.0 was used to edit and compile the source code [7,8]. BlueJ is a free, interactive Java environment developed and maintained by a joint group from Deakin University, and the University of Kent. The implementation code was completed in December 2007.

3 Example

Because of the page limitations of the paper, this section only presents the system converting an Object-Z specification to OO Perl; however, the system is fully capable of converting to both OO Perl and Java. The Object-Z class schema "text representation" in Figure 2 is not created until all data has been acquired through the GUI, however; it is presented first in order to give the reader a clearer picture of what will be defined. The class Object-Z schema example is called the Library class and within it, there are five elements in the visibility list, one Free Type, one UDT, one constant, one state variable, one initial state, and one public operation schema. Figure 2 illustrates the Library Object-Z Schema.

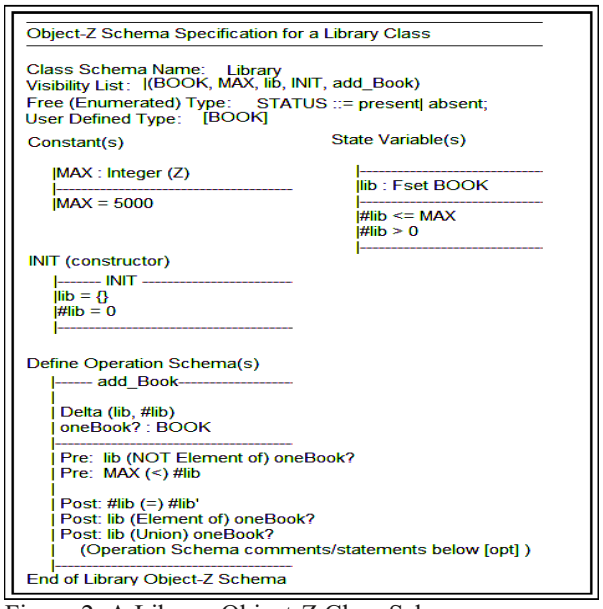


Figure 2: A Library Object-Z Class Schema

The user is able to select from a number of "Object-Z Class Schema" menu options; however, if a "CLASSNAME".XML document does not yet exist and the user does NOT choose to create a new Object-Z schema, valid data cannot be guaranteed. Figure 3 is the screen shot of the panel "Object-Z Class Invariant Schema" that assigns value to the class constants and specifies the class invariant for the system.

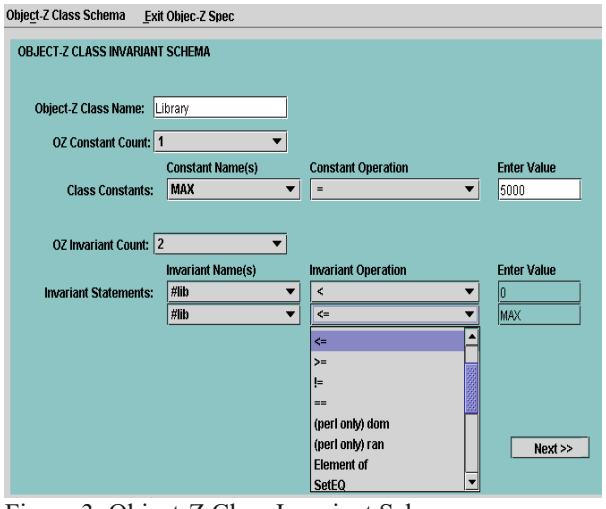


Figure 3: Object-Z Class Invariant Schema

Figure 4 illustrate a section of OO Perl skeletal code generated from the systems conversion of the Library class schema. The checkClassInvariant() subroutine (sub) is called at the beginning and end of every sub, including the constructor. The figure illustrates the variables, constants, constructor, the checkClassInvariant() and add_Book() subs. The checkClassInvariant() sub calls the boolean subs of the Assertions Class. The specific conditions are checked and the sub returns a value of true or false for each condition. The return values of all checked conditions are evaluated using logical ands, resulting in the final checkClassInvariant() result.

The add_Book() sub takes one parameter oneBook of type BOOK, and returns an int value. The returned value is -1 by default if either the pre- or the post- condition fails. The return values can also be modified by the programmer in the body of the method to make the values more meaningful.

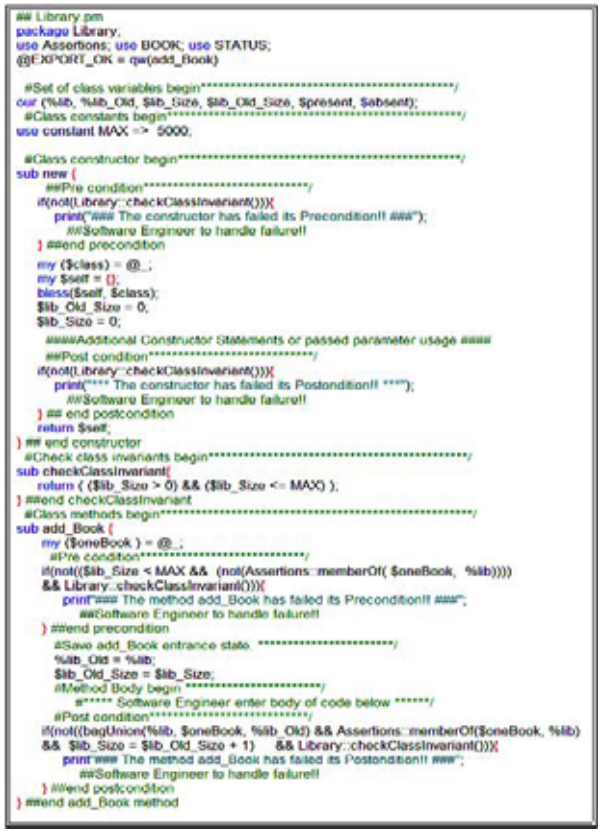


Figure 4: Perl Library Class, Variables and Constructor, Class Invariant and add_Book Method

Figure 5 depicts the Perl Assertions Class. This class is created for each new skeletal class that is generated. It contains subs that allow processing, within the generated skeletal classes by checkClassInvariant(), of set, sequence and bag operations. The Assertions class currently holds methods that process only a subset of all Object-Z operations. The methods "bagUnion" and "memberOf" are displayed in Figure 5.

Figure 5: PerlAssertions Class, Skeletal Code

4 Comparison and Future Work

This work is inspired by previous efforts in converting Object-Z specifications to object-oriented programming languages [1,2,13,14]. However, this work is aimed at adding design-by-contract written in logic assertions into Java and OO Perl and at bridging the semantic gap between formal specifications in Object-Z and design contracts in Java and OO Perl. The automatically generated skeletal code in Java and OO Perl provides programmers with flexibility in implementation details including the selection of algorithms. By building assertions of design contracts into programs and by guaranteeing the semantic consistence of design contracts between formal specification and implementation, more reliable and correct systems can be developed; thereby decreasing testing time and development cost while increasing system reliability and robustness.

As a significant extension to [13], the current "Object-Z to Java/OO Perl" system moves forward in this direction to translate formal specifications into implementation skeletal code by delivering a system with improved functionality. Compared with [13], which is limited to the provision of Object-Z primitive data types only, this project has succeeded in supporting a much larger subset of Object-Z basic types, compound types and their operations for conversion from specification to implementation. These specific types provided now consist of UDTs, free types, sets, bags, and sequences necessary for the

specification of realistic problems. The conversion rule set, which incorporates basic operations and a subset of compound operations, was made possible by enhancing the detail of the tags, represented in the XML DTD, and the data, collected by in the XML Manager. The XML DTD continues as the structure by which the specification's syntactic errors are caught and transferred to the user.

A second OO programming language, Per, was added for conversion in order to provide the user with additional options. The GUI offers improved menu options; file navigation directly to the directory containing the XML documents; and more detailed labeling. The system now provides a graphical representation of the Object-Z class schema, the user is given the option of opening, and editing any of the newly created documents, which include the new implementation skeletal class, any UDT skeletal class documents that may have been created, and the Object-Z class schema. The system also provides users with the opportunity to experiment with and gain a better understanding of formal methods and their usefulness.

There are still a number of ways to extend this system further. Currently, a statement is printed when an assertion fails that provides the user with information about what and where the assertion failed. By adding exception handling for the failed assertions, the program could require less user interaction and move more towards program automation.

Offering additional basic and compound operations like, existential and universal quantifiers, domain and range subtraction and restriction will help to improve the system. Creating a more generic framework that provide multiple language support such that the system is able to convert from a set of formal specifications to a set of implementation programming languages would produce wider acceptance and enhance the system further. A very useful addition would be a help menu along with meaningful comments to display detailed information and instructions about elements of the GUI panels.

This system uses the XML DTD as a means of modeling the XML document data. However, XML schema definitions (XSDs) provide greater functionality and flexibility when modeling the XML document data. The XSD supports a variety of data types, as well as uses more familiar XML elements and attributes, whereas XML DTD only supports strings or string lists and is stricter and less intuitive [11]. In future versions, using XSD can improve the expressiveness of the XML representation for this type of conversion.

References

[1] C. Fischer. (2000). "*Combination and implementation of processes and data: from CSP-OZ to Java*". PhD thesis. University of Oldenburg.

[2] C. Fischer. (1999). *"Software development with Object-Z, CSP and Java: A pragmatic link from formal specifications to programs*". Formal Techniques for Java Programs. Technical Report 251. Fernuniversität Hagen.

[3] M. Fukagawa, T. Hikita, and H. Yamazaki. (1994). "*A Mapping System from Object-Z to C++*".1st Asia-Pacific Software Engineering Conference (APSEC94). IEEE Computer Society Press.

[4] Griffiths. (1995). "*From Object-Z to Eiffel: a rigorous development method*". Technology of Object-Oriented Languages and Systems: TOOLS 18. Prentice Hall.

[5] Harry. (1996). "*Formal Methods Fact File VDM and Z*". John Wiley & Sons. Baffin Lane, Chichester. pg. 1-22, 173-292.

[6] W. Johnston and G. Rose. (1993). "*Guidelines for the Manual Conversion of Object-Z to C++*". SVRC Technical Report 93-14.

[7] M. Kölling and D. J. Barnes. (2006). "*Objects First with Java: A Practical Introduction using BlueJ*". 3rd Edition, Prentice Hall.

[8] M. Kölling . (2007). "*BlueJ – The Interactive Java Environment*". Teaching Java – Leaning Java. http://www.bluej.org/index.html

[9] Meyer. (1992). *Eiffel: The Language*. Prentice Hall.

[10] Meyer. (1997). *Object-Oriented Software Construction*. Prentice-Hall.

[11] M. Morrison. (2006). "*Sams Teach Yourself XML in 24 Hours*". 3rd edition. Sams Publishing.

[12] G.-H.B. Rafsanjani and S.J. Colwill. (1992). "*From Object-Z to C++: A Structural Mapping*". Z User Meeting (ZUM'92). Springer-Verlag.

[13] S. Ramkarthik, C. Zhang. (2006)."*Generating Java Skeletal Code with Design Contracts from Specifications in a Subset of Object Z*". 5th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2006). pp. 405-411.

[14] K. Rangarajan. (2000). "*Design by Contract for Java using JMSAssert".* A white paper document.

[15] G. Smith, (1999). "*The Object-Z Specification Language*". Kluwer Academic Publishers.

[16] J. Sun, J. S. Dong, J. Liu and H. Wang. (May 2001). "*Object-Z Web Environment and Projections to UML*". 10th International World Wide Web Conference (WWW-10). ACM Press. pages 725-734. Hong Kong.

[17] R. Wagner and R. Mansfield. (2003). "*XML All-In-One Desk Reference for Dummies*". Wiley Publishing. Inc., New York, NY

An Empirical Study on Modularization of Object Oriented Software *

Jing Liu[‡#], Bin Liu[†], Chi K. Tse[‡] and Keqing He[#]

[#]State Key Laboratory of Software Engineering, Wuhan Univ., Hubei, China
[†]Computer Science Dept. Wuhan Univ., Hubei, China
[‡]Dept. of Electronic and Information Engineering, The Hong Kong Polytechnic Univ., Hong Kong

Abstract

Object-oriented programming is supposed to produce better modularized structure than structural programming in that it encourages related state and behavior to be organized together in the form of classes, thus facilitating reuse and maintenance. To test whether classes in object-oriented software are well modularized, we conduct empirical studies on real world object-oriented software. By employing method from network analysis, object-oriented software structures are characterized as networks of methods. Metric and methods from community discovery research are applied to the analysis of modularization. From the empirical results, we conclude that not all object-oriented software classes are well modularized and there is a need for research on modularization improvement for classes of objected-oriented software.

1 Introduction

In 1972 Parnas first introduced information hiding as an approach to devising modular structures for software design. This approach had a tremendous impact on software industry. It contributed to the development of abstract data type programming languages, object-oriented design and programming, and the discipline of software architecture.

A fundamental approach to improving software development has been to modularize the design by splitting the implementation of the solution into parts [1]. Program parts can sometimes be termed modules. Modules often consist of data structures and one or more procedures/functions. In object-oriented programming paradigm in particular, modularization is realized by encapsulation, in which classes are used to encapsulate related variables and functions.

Object-oriented software is supposed to be better than structural programming in that it encourages organizing related state and behavior together in the form of classes, facilitating reuse and maintenance. Although much research has been devoted to automatic system reorganization in structural programming paradigm [2]–[4], and some research on grouping classes into subsystem in object-oriented software [5]–[6], there is very little done on improving the modularization of classes in object-oriented software. This might result from the presumption that classes in object-oriented software have well modularized structures.

In this paper, we test this presumption by conducting an empirical study on some object-oriented software. Software structures are characterized as networks, in which nodes represent methods, and edges represent interactions between methods. The modularizations of these networks are evaluated by a metric from the community discovery research, namely modularity.

Similar work was done by Lisa K. Ferrett etc. [19] who did an empirical comparison of modularity of procedural and object-oriented Software. Instead of comparing the inherent structure, however, their work focused on statistical values such as number of lines per module and number of parameters per module.

The organization of the paper is as follows: Section 2 lays the foundation of this paper by first giving a definition of method network, then illustrating the relationship between class and community. A concept from the community discovery community, modularity, is introduced and related with the software modularization. Section 3 presents the results from the empirical studies on real world software. Limitations of this work are discussed in Section 4. Finally, Section 5 concludes the paper and presents the possible work for future study.

*This work was supported by the Ph.D. Programs Foundation of Ministry of Education of China under grant No. 20070486065; the Natural Science Foundation of China under grant No. 90718005 and No.90604005; the National Basic Research Program of China (973) under grant No. 2007CB310800. and No. 2006CB708302.

2 Method Network, Class and Community

2.1 Definition of Method Network

Real-world software systems can be regarded as networks, in which software components, such as objects, classes, packages, subsystems or modules, are abstract nodes and the relationships (or interactions) between components are abstract edges [9]–[10].

In this paper, to study the modularization of classes, we use network structure to characterize software structure at method level, namely method network. In a method network, each node represents a method, and an edge represents the interactions between two methods.

Here we consider two kinds of interaction between methods. One is the method calling interaction. In this case, an edge is drawn between the "callee" and the "caller". The other kind of interaction happens between methods within the same class when the two methods refer to the same instance variable. For example, if both method A and method B refer to an instance variable c, then an edge is drawn between nodes A and B. Presently, no distinctions are made between these two kinds of interactions in the method network.

Definition The method network of an object-oriented software system is an undirected graph represented by $MN = (M, E)$. The set of nodes M corresponds to the methods of the software system, and the set of edges E represents the interactions between methods.

$Parent(x)$ is an operator returning the parent class of method x.

$InstanceVar(x)$ is an operator which returns the set of all the instance variables referred by method x.

$Call(x)$ is an operator returning the set of all the methods that method x calls.

Given two methods p, q, there are two corresponding vertices in the network $p, q \in M$.

If $Parent(p) = Parent(q)$ and $InstanceVar(p) \bigcap InstanceVar(q) \neq \phi$, then $(p, q) \in E$;

If $q \in Call(p)$, then $(p, q) \in E$.

Figure 1 gives a simple example of a method network.

2.2 Class and Community

Over the last decade, complex networks have been extensively studied within the mathematics, physics, biological science, nonlinear science, information science, and engineering communities [7]–[8]. Networks are used to characterize the structures of various systems. Methods from statistics, graph theory, etc. are applied in network analysis. As a result, many interesting phenomena are discovered, such as the scale-free and small-world characteristics of most real world network.

In the investigation of complex networks, identifying highly interconnected parts, namely communities (functionally related proteins, industrial sectors, groups of people, etc.), is crucial to the understanding of the structural and functional properties of various networks [11].

Community structure refers to the division of network nodes into groups within which the network connections are dense, but between which connections are sparser [12]. Community discovery techniques have been successfully applied to the discovery of interest groups in social network [13] and functionally related proteins in biological networks [14].

In object-oriented programming, classes are used to encapsulate variables and methods. It has the advantage of grouping together data and their operations. Methods within a well designed class should be closely related by the variables they work on. As required by the software design principle of low coupling and high cohesion, inherent interactions should be as intense as possible, while interactions between methods from different classes should be kept at a low level if not evitable. Therefore, when characterized by a network, methods within the same class could be regarded as a natural community.

The basic idea of our approach is to apply the community discovery algorithm to the method network extracted from the source code of the studied software. Methods divided by the classes they belong to formed a natural division of the network. If an object-oriented software has a well modularized structure, the natural division by classes should be similar to those obtained from the community discovery algorithm. Otherwise, we might want to reconsider the structure of the software under study.

2.3 Modularity

In the research of community discovery algorithms, it is often difficult to decide how many communities a network should be split into. Researchers found it necessary to set up a general criterion of how good a community division is.

Newman and Girvan [15] proposed that the divisions be evaluated using a measure known as modularity, which is a numerical index reflecting on how good a particular division is. For a division with g groups, we define a $g \times g$ matrix e whose component e_{ij} is the fraction of edges in the original network that connect vertices in group i to those in group j. Then the modularity is defined to be

$$Q = \sum_i e_{ij} - \sum_{ijk} e_{ij} e_{ki} = \mathrm{Tr}e - \|e^2\| \qquad (1)$$

$\|x\|$ indicates the sum of all elements of x. Physically, Q is the fraction of all edges that lie within communities

```
public class classX{
        public int i;     //Instance Variables
        private String str;  //Instance Variables
    private int methodA(double d, int num) {
                int j;
                j = d + num + i; //
                return j;
    }
    private String methodB(double d) {
                int j;
                String ret_str;
                j = d + i;
                ret_str = str + String.valueOf(j);
                return ret_str;
    }
    public void methodC(double d) {
        String cout = methodB(d);
        System.out.print(cout);
    }
}
```

```
public class ClassY{
        public double d;     //Instance Variables
        public int num ;
        public int outnum ;
    public void methodC() {
        classX x;
        x = new classX();
        x.methodA(d, num)
                return retmatrix;
        }
}
```

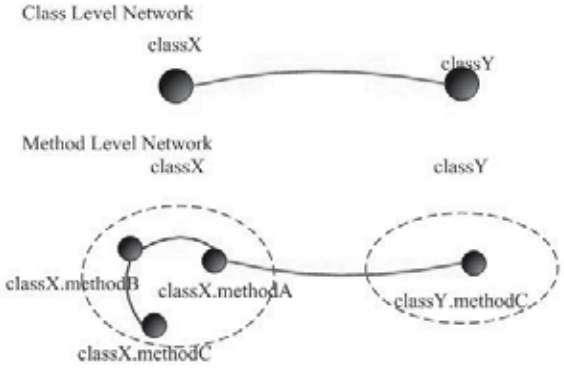


Figure 1. A simple example of Method network.

minus the expected value of the same quantity in a graph in which the vertices have the same degrees but edges are placed at random without regard for the communities.

If the network has no community structure, Q equals to 0. On the other hand, if the given network does have a community structure, the larger the values of Q are, the more accurate a partition is. If Q=1, which is the maximum, it indicates a strong community structure. In general, for typical real networks the value of Q falls between 0.3 and 0.7.

A good division should be one whose inner connection is intense while the external connection is sparse. It is the same feature we want for the software structure. Therefore, we can use modularity as an indication of the modularization of object-oriented software.

Similar quantitative measures, Modularization Quality (MQ) [20] and its many variants [21]–[23], were also proposed in automatic procedural program restructuring research. All based on the idea of measuring the extent of cohesion and coupling by comparing the number of the dependencies within and between subsystems.

3 Empirical Results

In this section, we will demonstrate how our approach is applied to real world object-oriented software analysis. We have chosen two open source software as the objects of our study. One is COLT which is an infrastructure for scalable scientific and technical computing in Java. It provides a set of Open Source Libraries for high performance scientific and technical computing in Java. The other one is JMetric, which is an object-oriented metrics analysis tool which supports metrics calculation for java programs.

They are deliberately chosen to represent two different categories of software. COLT is a library providing high performance computation utilities, therefore it emphasizes performance as well as reusability. JMetric represents the more general class of object-oriented applications.

Table 1 summarizes the general information of the two software.

Figure 2 demonstrates the size distribution of all the connected subgraphs that have at least 5 nodes. As can be seen from the Figure 2, there is only one connected subgraph of

Table 1. General information of the studied software.

	COLT	*JMetric*
Class Number	297	87
Method Number	3,141	1,738
Method Connections	20,413	212,936
Largest Connected Subgraph Size	1438	1411
Connected Subgraph Number(size¿5)	40	12
Avg. CBO	6.23	8.91
Avg. LCOM	0.47	0.51

size larger than 1000 in both COLT and JMetric. In JMetric, all the connected subgraphs are of size less than 25 except for the largest one. In COLT, however, there are 7 subgraphs of size around 100.

The JMetric has more than two hundred thousand edges at the level of method, which is more than ten times that of COLT. This suggests that the interactions in JMetric are very intense.

We calculate the CBO and LCOM metrics from the CK metrics suite [16]. CBO for a class is a count of the number of other classes to which it is coupled. The coupling between classes includes the inheritance, collaboration, and dependency relationships. LCOM value provides a measure of the relative disparate nature of methods in the class. Original definition of LCOM is the number of pairs of member functions without shared instance variables, minus the number of pairs of member functions with shared instance variables. And it arouses many critics. Here, LCOM is calculated according to the definition in [17]. It equals the percentage of methods that do not access a specific attribute averaged over all attributes in the class. A low value of LCOM indicates high cohesion and a well-designed class. These two values can give an overall idea of the coupling and cohesion degree of software

As suggested by the results, the two software show little difference in the average LCOM over classes. The average CBO of JMetric is larger than that of COLT, suggesting that JMetric has a higher degree of coupling. On the whole, they show little difference as far as the average CBO and the average LCOM values are concerned.

We compute the modularity of the natural community structure, namely the natural division of methods by the class they belong to. The modularity of the original partition of COLT is 0.759 compared with 0.113 of JMetric. As modularity is indication of modularization degree, the result suggests COLT and JMetric are very different from the view of modularization.

The high modularity value of 0.759 in COLT, indicates that the software is highly modularized and easy to be reused, which is in accordance with its specific aim of providing high performance computation library. Then we apply the community discovery algorithm[1] on its method network, and the modularity values of the generated partitions are computed. The modularity value of the generated partition of COLT rises from 0.759 to 0.789. The minor increase suggests that it would be difficult to make further improvements on the modular structure of COLT, and the software developers should consider the cost accompanied if they want to make any alterations.

Table 2. Result of Community Discovery on the largest connected subgraphs.

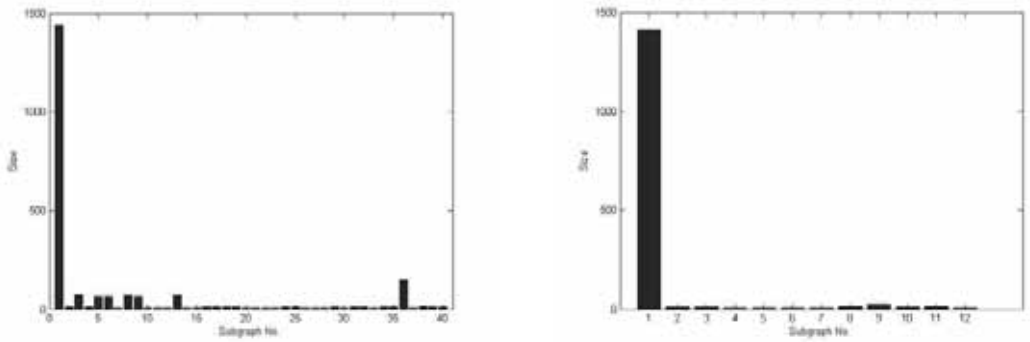
	Node	*Edge*	*Original Modularity*	*Generated Modularity*
COLT	1438	7437	0.759	0.789
JMetric	1411	109125	0.113	0.181
Azureus	7560	44926	0.434	0.679
Tomcat	7621	56803	0.447	0.594

The rather low value in modularity in JMetric, on the contrary, indicates the software is not well modularized, which is against the common belief that classes in object-oriented software have a well modularized structure. Again we apply community discovery algorithm and compute the modularity value of the generated communities. The value rises from 0.113 to 0.181, which is no great improvement either.

The minor improvement in JMetric, however, tells a different story. There should be something about the structure of JMetric which makes the division of the software difficult. Since the network is densely connected, we try to remove some of the inter community edges from the network. The inter-class method interactions can be resulted from three kinds of invocations according to the type of parent variable: local variable, method parameter or instance variable. We removed those method invocations whose parents are instance variables and computed modularity value. This time, the resulted modularity value is 0.808(compared with 0.0335 of COLT when the same process is carried). The great change in modularity value suggests that both the inner and the outer connections of JMetric classes are very dense. The software is very compact and it would be difficult if we just want to reuse part of it.

Further experiments on other object-oriented software, such as Azureus and Tomcat, demonstrate medium level of modularity values. Large increases of the modularity val-

[1]Community Discovery algorithm used here is Local Community Detecting Algorithm (LCDA)[18]. Since community structure tends to interiorly densely connected while sparsely connected externally, the main idea of LCDA is to use clustering coefficient as a criterion to determine the community structure., Only when its clustering coefficient is more than a predetermined threshold value, can the community be a member of the community structure.



(a) COLT (b) JMetric

Figure 2. Subgraph size of COLT and JMetric.

ues are witnessed in the generated communities, suggesting further improvement on the structure modularization is possible.

4 Threats to Validity

Threat to validity of this experiment can result from the following three aspects:

- Only regular methods are considered, inherited methods are not taken into account.
- Not all sorts of coupling and cohesion are considered. Only two kinds of interactions between methods are considered: method call and sharing of variables. These interactions can cover communicational cohesion,functional cohesion within classes, and only some of the dependency and collaboration couplings between classes. However, there are certain coupling and cohesion left out, for example, sequential cohesion, data coupling or stamp coupling.
- The definition of good modular structure in Object Oriented software can be controversial. For example, it is viable, cohesive way to group related methods without data. However, it is not cohesive in the "connected via data" sense.

However, this research should be seen as exploratory, and while these threats exist, it does not prevent us from using the research as a basis for future studies.

5 Conclusion and Future Work

In this paper, we perform empirical study on some object-oriented software to test whether they are well modularized at the level of Class.

By employing method for network analysis, we characterize an object-oriented software structure as a method network. And Classes are regarded as natural division on the methods, or natural communities. The modularity value, a metric for measuring the quality of a given partition of communities, is used to measure the modularization extent of software.

Several object-oriented software are studied. Two are specifically discussed, COLT and JMetric. Based on the results from the empirical studies, we make the following observations:

- Not all object-oriented software classes are well modularized.
- Modularity value can be used as an indication of the modularization level, providing software developers with a guideline for software reusability and structural improvements.
- Research should be directed towards improving modularization of classes in some objected-oriented software.
- The community discovery algorithm might be useful in providing software structural improvement.

In the future, we plan to apply the community discovery algorithms to method networks to improve modularization of the classes in object-oriented software.

References

[1] Darcy, D.P., et al., "The Structural Complexity of Software: An Experimental Test," *IEEE Transactions on Software Engineering* 2005. 31(11): p. 982-995.

[2] Hutchens, D.H. and V.R. Basili, "System structure analysis: clustering with data bindings," *IEEE Transactions on Software Engineering*, 1985. 11(8): p. 749-757.

[3] Schwanke, R.W., "An intelligent tool for re-engineering software modularity," *Proc. of 13th International Conference on Software Engineering*. 1991. p. 83-92.

[4] Muller, H.A., et al., "A Reverse Engineering Approach To Subsystem Structure Identification," *Practice*, 1993. 5(4): p. 181-204.

[5] Seng, O., et al.,"Search-based improvement of subsystem decompositions", *Proc. of the 2005 conference on Genetic and evolutionary computation*, 2005. p. 1045-1051.

[6] Wen, Z. and V. Tzerpos, "Software clustering based on omnipresent object detection," *Proc. of 13th International Workshop on Program Comprehension*, 2005. p. 269-278.

[7] Barabási, A.L. and R. Albert, "Emergence of Scaling in Random Networks," *Science*, 1999. 286(5439): p. 509-512.

[8] Watts, D.J. and S.H. Strogatz, "Collective dynamics of'small-world'networks," *Nature*, 1998. 393(6684): p. 409-10.

[9] Myers, C.R., "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs," *Physical Review E*, 2003. 68(4): p. 46116.

[10] Liu Jing, H.K., Ma Yutao, Peng Rong, "Scale Free in Software Metrics," *Proc. of IEEE Computer Software and Applications Conference*, 2006. p. 229-235.

[11] Palla, G., et al., "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, 2005. 435(7043): p. 814-818.

[12] Newman, M.E.J., "Detecting community structure in networks," *Eur. Phys. J. B* 38, 321C330 (2004).

[13] Girvan, M. and M.E.J. Newman, "Community structure in social and biological networks," *Proc. of the National Academy of Sciences*, 2002. 99(12): p. 7821-7826.

[14] Holme, P., M. Huss, and H. Jeong, "Subnetwork hierarchies of biochemical pathways," *Bioinformatics*, 2003. 19(4): p. 532-538.

[15] Newman, M.E.J. and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, 2004. 69(2): p. 26113.

[16] Chidamber, S.R., C.F. Kemerer, and C. Mit, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, 1994. 20(6): p. 476-493.

[17] B Henderson-Sellers, LL Constantine, IM Graham, "Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design) - Object Oriented Systems," *Object-Oriented Systems*, 3(3), pp143-158, 1996.

[18] Juan Liu, B.L., Deyi Li, "Discovering Protein Complexes from Protein-Protein Interaction Data by Local Cluster Detecting Algorithm," *Proc. Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, Aug. 2007.Vol. 4, pp. 280-284.

[19] Lisa K. Ferrett, Jeff Offutt. "An Empirical Comparison of Modularity of Procedural and Object-oriented Software," *Proc. of the Eighth IEEE international Conference on Engineering of Complex Computer Systems*,2002. p.173-182

[20] Mancoridis, S., et al., "Using automatic clustering to produce high-level system organizations of source code," *Proc. of 6th International Workshop on Program Comprehension*, 1998. p. 45-52.

[21] Mahdavi, K., M. Harman, and R.M. Hierons, "A multiple hill climbing approach to software module clustering," *Proc. of International Conference on Software Maintenance*, 2003. p. 315-324.

[22] Shokoufandeh, A., et al.," Spectral and meta-heuristic algorithms for software clustering," *The Journal of Systems & Software*, 2005. 77(3): p. 213-223.

[23] Harman, M., S. Swift, and K. Mahdavi, "An empirical study of the robustness of two module clustering fitness functions," *Proc. of 2005 Conference on Genetic and Evolutionary Computation*, 2005. p. 1029-1036.

Bridging the gap between slicing and model-based diagnosis*

Franz Wotawa
Technische Universität Graz
Institute for Software Technology
8010 Graz, Inffeldgasse 16b/2, Austria
wotawa@ist.tugraz.at

Abstract

Fault localization is considered an important and difficult task in the software engineering process. In the last decades several approaches to fault localization have been published. Some of them are based on either static or dynamic program slicing. In this paper, we present an approach that combines program slicing with the computation of hitting sets. Hitting sets are used in model-based diagnosis to compute diagnoses from conflicting assumptions. We introduce the underlying definitions and algorithms of the approach, and show that the combination of slicing and hitting set computation reduces the number of statement to be considered. The presented approach does not rely on a specific slicing methodology and can be used in combination with static or dynamic slicing.

1. Introduction

Debugging which comprises the activities fault detection, localization, and repair has been an active research area for the past decades. Although most of the research activities can be classified as activities regarding fault detection like formal verification or testing, some effort has been spent in providing tools for fault localization and even less for repair. In this paper, we focus on fault localization and present an approach that combines slicing techniques with the computation of hitting sets, a technique that originates from model-based diagnosis.

Program slicing was introduced by Mark Weiser [15, 16]. He argued that programmers use data-flow and control-flow dependences, and finally slices in order to focus their attention to the more important statements within the program in case of incorrect outputs. Mark Weiser took this observation and introduced the concept of static program slices. In [15, 16] a slice is a program where zero or more statements are removed, and which behaves in the same way as the original program for the specified variables at a given location in the program. This definition can hardly be directly implemented because it requires checking program equivalence. Hence, Weiser introduced an approximation algorithm for computing slices in a static way, i.e., only considering the program source code and no dynamic information. Because of the static analysis the Weiser-style slices tend to be larger than necessary for a failure revealing test case.

In order to make slices as small as possible but without losing precision, several improvements have been reported. Some include the use of information about correct program runs. One example is program dicing where the set difference between a static slice for a failure-revealing test-case and the static slice for a program run leading correct outputs is computed. Shahmehri et al. [13] pointed out that dicing is only correct with respect to some very restrictive assumptions. Other improvements and extensions for static slicing have been introduced because of the integration of programming language constructs like procedure calls or concurrency. Horwitz et al. [6] introduced an algorithm for computing the system dependence graph that is an extension of the program dependence graph [2] where procedure calls can be represented. Static slices can be easily computed from such graphs via graph traversal. Krinke [10] improved slicing of programs with procedure calls and extended slicing to handle concurrent programs. Although, those improvements lead to more precise static slices for general programming languages, the use of static slices for debugging is still limited because of their size.

To overcome this problem the concept of dynamic slicing was introduced by Korel et al. [9]. Dynamic slicing additionally takes care of the program execution and, therefore, usually results in smaller slices. But unfortunately dynamic program slices might not include the faulty state-

*The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsfrderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

```
1.  main(int argc, char *argv[])
2.  {
3.    int red, green, blue, yellow;
4.    int sweet, sour, salty, bitter;
5.    int i;
6.
7.    red = atoi(argv[1]);
8.    blue = atoi(argv[2]);
9.    green = atoi(argv[3]);
10.   yellow = atoi(argv[4]);
11.
12.   red = 2*red; //Error:red = 5*red;
13.   sweet = red*green;
14.   sour = 0;
15.   i = 0;
16.   while (i < red) {
17.     sour = sour + green;
18.     i = i + 1;
19.   }
20.   salty = blue + yellow;
21.   yellow = sour + 1;
22.   bitter = yellow + green;
23.
24.   printf("%d %d %d %d \n",
            bitter, sweet, sour, salty);
25.   return 0;
26. }
```

Figure 1. An example program taken from [5]

ments and several extensions like Critical Slicing [1], which combines program mutations and dynamic slicing, Relevant Slicing [19], which introduces a potentially depends relation for the same purpose, and Failure-inducing chops [5], which combine delta debugging [18] with slicing, have been reported. Again, precision and improving the size of slices have been the major driving force of this research. For an overview on slicing we refer the reader to Kamkar [7] or Tip [14]. Other applications of slicing to program debugging include work by Kamkar [8]. Most recently Kusumoto et al. [11] reported on the usefullness of slicing for debugging. In the paper the authors present an empirical study, which shows that programmers are more effective when using slices for debugging.

In this paper, we continue research on improving slicing for debugging. The presented approach potentially leads to smaller slices and is not restricted to a specific slicing methodology. Hence, every technique for computing slices can be used. Before formalizing the approach, we first give an example. The program we are using is given in Figure 1 and was used by Gupta et al. [5] in their paper. The input `[1,5,8,2]` forms a failure-revealing test case. The outputs `bitter`, `sweet`, and `sour` at line 24 are incorrect. For each incorrect output at line 24, we compute a dynamic slice:

- `bitter`: {7,9,12,14,15,16,17,18,21,22,24}
- `sweet`: {7,9,12,13,24}
- `sour`: {7,9,12,14,15,16,17,18,24}

Informally, the semantics of a slice for a slicing criterion comprising a variable at a position and a test case can be stated as the set of statements where each statement contributes to the incorrect computation. In this case we know that at least one of the statements of the slice is responsible for the observed behavior. This observation is equivalent to the following sentence: It is a contradiction to assume that all statements of a slice are correct. Such contradictions are also called conflicts in model-based diagnosis [12].

In order to eliminate all conflicts, we have to take one necessarily not different element from each conflict, i.e., slice, and assume that the element, i.e., statement, behaves not correct. When doing so we eliminate all possible conflicting assumptions. The selected elements have an intersection with every slice and are called hitting sets. In model-based diagnosis the hitting sets of conflicts are diagnoses. In most cases someone is interested in small diagnoses with respect to their size. For our example, we obtain 4 diagnoses of size 1, i.e., 7, 9, 12, and 24, because these statements are elements of each slice. If we are only interested in single faults, then computing the intersection of all conflicts would be sufficient. However, in case of multiple faults the intersection of conflicts can be empty. Therefore, a general approach cannot rely on intersection.

The paper is organized as follows. In the next section we start with the basic definitions and give an algorithm for computing hitting sets. Afterwards, we introduce the approach and present an algorithm which combines hitting sets with slices. We further present a small case study and an extension that handles the closer integration of slicing and hitting set computation, and finally we conclude the paper.

2. Hitting sets

Model-based diagnosis [12] is a troubleshooting methodology, which allows computing explanations for a certainly detected misbehavior directly from the model. Explanations are called diagnoses. Diagnoses are computed from conflicts, i.e., parts of the model, which lead to an inconsistency considering the given observations. For this purpose we use hitting sets, which we define in this section of the paper in order to be self contained. For the formal definitions and an algorithm we refer the reader to Reiter's

work [12]. Greiner et al. [4] presented a corrected version of Reiter's algorithm.

Hitting sets are defined over a set of sets with the property that the intersection of a hitting set with every given set is not empty.

Definition 1 (Hitting set) *A set $\Delta \subseteq \bigcup_{x \in F} x$ for a set of sets F is a hitting set iff the intersection of Δ with all elements of F is not empty, i.e., $\forall x \in F : \Delta \cap x \neq \emptyset$.*

A hitting set is minimal if no proper subset of a hitting set is itself a hitting set. Usually we are only interested in minimal hitting sets and if not otherwise mentioned we always refer to minimal hitting sets when using the term hitting set. Note that the definition of minimal hitting set is not based on cardinality. For example the set $\{12\}$ is a minimal hitting set for $\{\{12,13,15\}, \{12,14,16\}, \{12,14\}\}$ but also $\{13,14\}$.

Reiter [12] introduced an algorithm for computing hitting sets that has been improved later by Greiner et al. [4]. The algorithm uses the given sets of sets F and constructs a directed acyclic graph (DAG) in a breadth first manner. After the construction of the DAG the minimal hitting sets correspond to some vertices of the DAG which are labeled with a $\surd$. The algorithm needs not to compute all possible hitting sets. Instead the user can specify the maximum cardinality of the obtained hitting sets. For practical applications especially in cases where the size of the input is large, such a boundary value is of great use. The following algorithm is a variant of the original algorithm where we eliminated one pruning rule. This elimination is possible when assuming that F is a sorted collection with respect to the cardinality of the sets where the left-most element has the smallest one.

Algorithm Hitting-Set-Computation
Input: A sorted collection F of sets with respect to cardinality. The smallest set is assumed to be the left-most element of F. A number $MAX > 0$ which specifies the maximum size of the generated hitting sets.
Output: All minimal hitting sets of F.

1. Let H be the growing DAG and L be the empty set. Generate a new node n, which is the root node of H, add it to H, let $label(n)$ and $h(n)$ be the empty set. Add n to L, let L' be the empty set and set $i = 0$.

2. For all nodes n in L do:

   (a) From left to right search for a set $C \in F$ such that $C \cap h(n)$ is the empty set. If there is no such set, a new minimal hitting set has been found and let $label(n) = \surd$.

   (b) Otherwise, for each $x \in C$ do:

      i. If there exists a previously handled node m with $h(m) = h(n) \cup x$, then generate a new arc from n to m.

      ii. Otherwise, generate a new node n' with $h(n') = h(n) \cup x$, and an arc from n to n'. If there exists a previously handled node m with $label(m) = \surd$, and $h(m) \subset h(n')$, then close node n' and let $label(n') = \times$. Otherwise, add n' to L'.

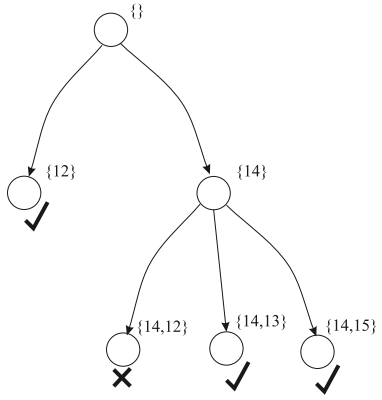
   (c) Let $i = i + 1$.

3. If L' is not empty, and $i \leq MAX$, let L be L' and $L' = \emptyset$, and go to 2.

4. Otherwise, return a set comprising $h(n)$ for all nodes n with $label(n) = \surd$.

The hitting set algorithm obviously terminates for every finite set F and MAX. If F is not empty, then the algorithm has at least two iterations. All hitting sets can be computed by setting MAX to the number of elements stored in F's set, i.e., $MAX = |\bigcup_{x \in F} x|$.

The hitting set DAG for F =$\{\{12,13,15\}, \{12,14,16\}, \{12,14\}\}$ is given as follows where the values of h for each node are given under parentheses ($\{\}$):



Note that the algorithm has to be called on the sorted collection $F' = \{\{12, 14\}, \{12, 14, 16\}, \{12, 13, 15\}\}$ and not on the original set F. Hence, we finally obtain 3 minimal hitting sets.

3. Hitting sets and slices

In this section, we formally introduce our fault localization process. We assume a program Π that is written in a programming language $\mathcal{L}$. We further assume a semantics of $\mathcal{L}$ defined by a function $[\![]\!] : \mathcal{L} \times \Sigma \mapsto \Sigma$ which maps programs and states to new states. In this definition a state $s \in \Sigma$ specifies values for variables used in the program.

We further assume that the program Π is correct with respect to the grammar of $\mathcal{L}$ and halts on every given input. A test case is a tuple (I, O) where $I \in \Sigma$ is the input and $O \in \Sigma$ is the expected output. A program Π passes a test case $t = (I, O)$ iff $[\![\Pi]\!]I \supseteq O$. Otherwise, we say that the program fails. Because of the use of the $\supseteq$ operator also partial test-cases are allowed which do not specify values for all output variables. If a program passes a test case t, then t is called a positive test case. Otherwise, the test case is said to be a negative test case. Note that we do not consider inconclusive test cases explicitly. In cases where inconclusive test cases exist, we treat them like being positive test cases. Since we are only considering negative test cases for fault localization this assumption has no influence on the final result. A test suite TS for a program Π is a set of test cases and can be partitioned into two disjoint sets comprising only positive (POS) respectively negative (NEG) test cases, i.e., $TS = POS \cup NEG \wedge POS \cap NEG = \emptyset$.

For a negative test case $t = (I, O) \in NEG$ we know because of the definition that there must be some variables $CV_t = \{x_1, \ldots, x_k\}$ where for all $i \in \{1, \ldots, k\}$ $x_i = v_i \in O$ and $x_i = w_i \in [\![\Pi]\!]I$ follows that $v_i \neq w_i$. We call such variables $x_1, \ldots, x_k$ conflicting variables. For each of the variables $x_1, \ldots, x_k$ we compute a slice $S(x_1), \ldots, S(x_k)$ as follows: $S(x_i) = SLICE(\Pi, \langle t, n, \{x_i\}\rangle)$ where $SLICE$ is a function implementing the computation of either static or dynamic slices, t is a test case, n is the line of the program where variable x_i is known to hold the wrong value. Note that we have no restrictions on the computation of slices but using slicing algorithms that are incorrect or produce imprecise results will cause our approach to compute itself incorrect or imprecise results. The corresponding conflict set for a negative test case t is now given as $C_t = \{S(x)|x \in CV_t\}$. From this conflict set we compute all minimal diagnosis, e.g., $DIAGS_t = \{\Delta|\Delta \in HS(C_t)\}$ where HS implements the introduced hitting set algorithms for the given set C_t.

Before discussing some implications of the above definitions we illustrate the approach using our example program from Fig. 1. Given a test cases that is described in the introduction we obtain the following sorted collection of conflicts when using dynamic slicing: F ={{7,9,12,13,24}, {7,9,12,14,15,16,17,18,24}, {7,9,12,14,15,16,17,18,21,22,24}}. From this collection our implementation of the hitting set algorithms computes 9 diagnoses within a fraction of a second: {7},{9},{12},{24} are single fault diagnoses and {13,14}, {13,15}, {13,16}, {13,17}, {13,18} are double fault diagnoses.

When having a look at the obtained theory and results someone might ask why not using the intersection of conflicts directly instead of computing minimal diagnoses using a hitting set algorithm? To answer this question, we first define the intersection formally as $INTERSECT_t = \{\{n\}|n \in \bigcap_{x \in C_t} x\}$. From this definition follows that for every set $\{i\} \in INTERSECT_t$, i itself has to be an element of every conflict. Accordingly to the definition of hitting sets and diagnosis, $\{i\}$ would also be element of $DIAGS_t$. Hence, all elements of $INTERSECT_t$ are single fault diagnosis of $DIAGS_t$. But in cases where $INTERSECT_t$ is empty, $DIAGS_t$ still provide usefull information, e.g., that there are no single fault diagnoses. The same resutls cannot be obtained when using the intersection operator. Therefore, we conclude that the hitting set computation is more general than computing the intersection only. The following corollary summarizes these findings.

Corollary 1 *Given a program* Π *and a negative test case* t*. The intersection* $INTERSECT_t$ *of all conflict sets for* t *given* Π *is a subset of the set of all diagnosis* $DIAGS_t$ *for the same conflict set. If* $INTERSECT_t$ *is the empty set, than all elements of* $DIAGS_t$ *have a size greater than 1, i.e., in this case there are only multiple fault explanations for a negative test case.*

Our diagnosis approach only delivers better results when there are more different slices. In cases where only one slice is available because only one output contradicts the expected output values of a test case, every element of the slice is a minimal single fault diagnosis. Hence, the approach does not gain new information in this case.

Corollary 2 *Given a program* Π *and a negative test case* t*. If the set of contradicting variables contains only one element* x*, then all elements of the corresponding slice* $SLICE(\Pi, \langle ll(\Pi), \{x\}\rangle)$ *are single fault diagnosis.*

Note that the above process makes only use of a single negative test case. It is of course also possible to include all slices coming from all negative test cases for a particular program comprising the same input and output variables. In this case depending on the construction of the test cases multiple faults becomes more likely.

4. Case study

In order to further evaluate the capabilities of the presented approach, we started with a case study. This study includes 5 small programs ranging from 12 to 129 lines of code. All of the used examples have several inputs and several outputs. All programs except the first one are implementing digital circuits. The first one is the one from Fig. 1. Faults were introduced in the program manually. Test cases were computed randomly using the original programs as specifications. During random test case generation a lot of failure-revealing test cases, which make more than

1 output incorrect, could be obtained. From the obtained slices we computed the diagnoses using a Java implementation of the introduced hitting set algorithm. In all cases the diagnoses could be obtained within less than a second on a standard PC.

The statistical information regarding the considered programs as well as the obtained results are given in Table 1. `example` is the program from Fig. 1. `alu` is a Java implementation of an arithmetic logic unit. `4_bit_adder` is a Java implementation of a binary 4 bit adder. `c17` is a Java implementation of a ISCAS85 circuit. The ISCAS85 suite is used as benchmark suite in the hardware design community. Finally, `code_converter` implements a seven segment code conversion device.

Table 1 gives the lines of code (LOC), the number of slices, the minimum (Min) and maximum (Max) size of the slices, the size of the union of all obtained slices (All), the number of single fault diagnoses (Bugs), where the set of bugs is the result of the hitting set computation, and the percentage of single fault diagnoses with respect to the lines of code. In all examples the number of single fault diagnoses is smaller than the smallest slice. If we build the union of the slices for each incorrect output, the reduction would be about 50 percent. When using the introduced approach, the reduction is between 80 to 95 percent. This means that in the best case only the remaining 5 percent of the source code has to be considered for further investigation during debugging.

5. Extensions

It has been shown in various papers, e.g., [15, 16] and [11], that programmers effectively make use of slices during fault localization. Although, the introduced notation of diagnosis lead to improvements in terms of a reduction of statements to be considered, it might not work as expected in all situations. Consider for example the case where only multiple fault diagnoses are available. A programmer might gain important information from the diagnoses. But the listed diagnoses do not allow for providing an overview. Only diagnosis after diagnosis can be looked at during the whole debugging process. If considering our example program in Fig. 1, we have to analyze the 5 double fault diagnoses one by one.

In order to overcome this problem, we describe how to map back diagnoses to some sort of summary slices. For this purpose, we enhance the information provided by a slice with a probability value that is assigned to each element of the slice. We call such a slice comprising statements and corresponding probabilities a HS-slice.

Let $DIAGS_t$ be the set of diagnosis obtained from a set of slices F for a negativ test case t, and a program Π using the hitting set algorithm. For each diagnosis $\Delta \in DIAGS_t$ we compute its probability. This probability is equivalent to the probability of the state that all elements in Δ are incorrect and that all other statements are correct. Formally, this probability is stated as follows (when assuming independence of failure):

$p(\Delta) = \prod_{s \in \Delta} p_F(s) \cdot \prod_{s' \in \Pi \setminus \Delta} (1 - p_F(s'))$

The fault probability of a statement s, i.e., $p_F(s)$, is usually not given. In such cases the assumption that all statements fail with equal probability is used. Using this assumptions the fault probability of statement s becomes $p_F(s) = 1/|\Pi|$, where $|\Pi|$ denotes the number of statements of program Π. Because the same probability applies for all statements, we drop $p_F(s)$ and use p_F from here on instead. We finally obtain the fault probability of a diagnosis Δ:

$p(\Delta) = p_F^{|\Delta|} \cdot (1 - p_F)^{|\Pi \setminus \Delta|}$

The probability of a statement s can be obtained by computing the sum of the fault probabilities of the diagnoses where the statement is an element.

$p(s) = \sum_{\Delta \in DIAGS_t \wedge s \in \Delta} p(\Delta)$

We now have all necessary pieces to define HS-slices.

Definition 2 (HS-slice) *Given a program Π, a test case t. A HS-slice S is a set of pairs that is obtained from the set of diagnoses $DIAGS_t$ as follows:*

$S = \{(s, p(s)) | \exists \Delta \in DIAGS_t : s \in \Delta\}$

Using this definition we obtain the following HS-slice for the example program from Fig. 1:

{(7,0.0139), (9,0.0139), (12,0.0139), (13,0.0027), (14,0.0005), (15,0.0005), (16,0.0005), (17,0.0005), (18,0.0005), (24,0.0139)}

Note that this slice is also smaller than the union of the three slices. It specifically indicates the statements with the highest fault probability. The fact that every double fault diagnoses has to have statement 13 as an element is also represented in an appropriate way.

6. Conclusion

The application of model-based diagnosis to software debugging is not new. Friedrich and colleagues [3] used model-based diagnosis together with a dependence-based model to localize bugs in VHDL programs. Wotawa [17] proved that the strong relationship between static slicing and model-based debugging using dependence-based models. In this paper, we present a more general approach for the integration of model-based debugging and slicing. We presented a first case study and finally an extension which allows for an easy integration with existing slicing-based approaches.

Because the approach is based on existing slicing techniques, the overall outcome depends on those techniques.

Table 1. Diagnosis results

Program	LOC	Inputs Slices	Outputs Min	No of Max	Size of Slices All			Bugs	Percentage
example	4	4	26	3	5	11	12	4	15.4
alu	14	8	129	4	32	57	65	25	19.4
4_bit_adder	8	5	56	4	6	13	25	3	5.4
c17	5	2	12	2	3	5	6	2	16.7
code_converter	6	7	68	5	10	13	30	7	10.3

Limitations of used slicing techniques will also be limitations of the approach. In cases where only one output is incorrect and, therefore, where only one slice is available, the approach does not improve the final outcome. This is not a severe problem because the computational requirements are not very demanding especially in the case of only one slice.

References

[1] Richard A. DeMillo, Hsin Pan, and Eugene H. Spafford. Critical slicing for software fault localization. In *International Symposium on Software Testing and Analysis (ISSTA)*, pages 121–134, 1996.

[2] Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, 1987.

[3] Gerhard Friedrich, Markus Stumptner, and Franz Wotawa. Model-based diagnosis of hardware designs. *Artificial Intelligence*, 111(2):3–39, July 1999.

[4] Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.

[5] Neelam Gupta, Haifeng He, Xiangyu Zhang, and Rajiv Gupta. Locating faulty code using failure-inducing chops. In *Automated Software Engineering (ASE)*, pages 263–272, November 2005.

[6] Susan Horwitz, Thomas Reps, and David Binkley. Interprocedural Slicing Using Dependency Graphs. In *Proceedings of the SIGPLAN'88 Conference on Programming Language Design and Implementation*, pages 35–46, Atlanta, Georgia, 1988.

[7] Mariam Kamkar. An overview and comparative classification of program slicing techniques. *J. Systems Software*, 31:197–214, 1995.

[8] Mariam Kamkar. Application of program slicing in algorithmic debugging. *Information and Software Technology*, 40:637–645, 1998.

[9] Bogdan Korel and Janusz Laski. Dynamic Program Slicing. *Information Processing Letters*, 29:155–163, 1988.

[10] Jens Krinke. Advanced slicing of sequential and concurrent programs. In *20th International Conference on Software Maintenance*. IEEE, 2004.

[11] Shinji Kusumoto, Akira Nishimatsu, Keisuke Nishie, and Katsuro Inoue. Experimental evaluation of program slicing for fault localization. *Empirical Software Engineering*, 7:49–76, 2002.

[12] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[13] Nahid Shahmehri, Mariam Kamkar, and Peter Fritzson. Usability criteria for automated debugging systems. *J. Systems Software*, 31:55–70, 1995.

[14] Frank Tip. A Survey of Program Slicing Techniques. *Journal of Programming Languages*, 3(3):121–189, September 1995.

[15] Mark Weiser. Programmers use slices when debugging. *Communications of the ACM*, 25(7):446–452, July 1982.

[16] Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, July 1984.

[17] Franz Wotawa. On the Relationship between Model-Based Debugging and Program Slicing. *Artificial Intelligence*, 135(1–2):124–143, 2002.

[18] Andreas Zeller and Ralf Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering*, 28(2), feb 2002.

[19] Xiangyu Zhang, Haifeng He, Neelam Gupta, and Rajiv Gupta. Experimental evaluation of using dynamic slices for fault localization. In *Sixth International Symposium on Automated & Analysis-Driven Debugging (AADEBUG)*, pages 33–42, 2005.

Dynamic Analysis and Design Pattern Detection in Java Programs

Lei Hu and Kamran Sartipi
Dept. Computing and Software, McMaster University, Hamilton, ON, L8S 4K1, Canada
{*hu14, sartipi*}*@mcmaster.ca*

Abstract

Identifying design patterns within an existing software system can support understandability and reuse of the system's core functionality. In this context, incorporating behavioral features into the design pattern recovery would enhance the scalability of the process. The main advantage of the new approach in this paper over the existing approaches is incorporating dynamic analysis and feature localization in source code. This allows us to perform a goal-driven design pattern detection and focus ourselves on patterns that implement specific software functionality, as opposed to conducting a general pattern detection which is susceptible to high complexity problem. Using a new pattern description language and a matching process we identify the instances of these patterns within the obtained classes and interactions. We use a two-phase matching process: i) an approximate matching of class attributes generates a list of candidate patterns; and ii) a structural matching of classes identifies exact matched patterns. One target application domain can be software product line which emphasizes on reusing core software artifacts to construct a reference architecture for several similar products. Finally, we present the result of a case study.

KEYWORDS: Dynamic Analysis; Design Pattern Detection; Feature-specific Scenario; Pattern Matching; Software Family; Data Mining.

1. Introduction

Software companies that satisfy the needs of a specific market segment develop products that share common sets of features [8]. These products are usually developed based on a reference architecture which consists of common parts and variable parts, where the variable parts can be modified to satisfy the evolving requirements of the new products. In this context, the evolutionary development of a software system starts from identifying the important features contained in similar products as well as identifying the reusable components based on the reference architecture [7].

In this paper, we propose a new approach based on a hybrid dynamic and static analysis to address the problem of reusing existing system's design patterns that correspond to specific software behavior as the goals of the recovery process. In this context, design patterns (i.e., common solutions to recurring design problems [11]) can assist a software engineer in comprehending and reusing design decisions and solutions adopted by the original software designers. Consequently, these patterns can be used in developing a family of similar systems that share the same core features.

The proposed framework identifies the existing design patterns in the key software features through two major parts: *dynamic analysis* and *pattern detection*. In dynamic analysis, we identify a group of key features of the subject system and generate a set of relevant task scenarios for each feature, namely feature-specific scenario set. Through scenario execution, pattern mining, and concept lattice analysis we obtain the classes that contribute in generating those features without any prior knowledge about the system. The obtained classes will form a search space to conduct the pattern detection process, where the design patterns are specified using a new pattern description language (PDL) that drives the pattern matching process. A pattern repository holds the specification of a number of design patterns. The pattern matching process recovers the instances of the design patterns in the repository in two phases: i) an approximate matching process generates a list of potential pattern instances for each target pattern, by comparing the number of class attributes in the search space; and ii) a structural matching compares the complete class structure of the target pattern against the structure of the candidate instance pattern.

In order to extract the core functionality of the existing systems the software solution providers need a set of diverse reverse engineering tools to be used for different projects and at different application domains [6]. The approach proposed in this paper contributes in such problem domain by the followings:

i) mapping software behavior to source code as a means to identify core classes that implement the key features of a software system; hence providing a reduced search

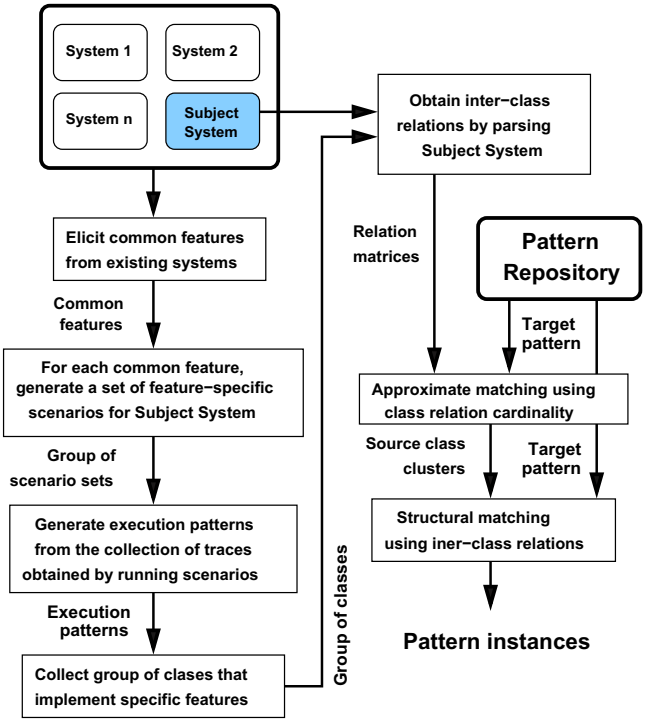


Figure 1. The proposed framework for dynamic analysis and design pattern recovery.

space for design pattern recovery; and

ii) presenting a novel two-phase search technique and a pattern definition language to perform design pattern recovery.

2. Related work

In this section, we discuss relevant approaches in dynamic analysis and design pattern detection to our work.

In dynamic analysis of software systems, El-Ramly et al. [10] applied a sequential pattern mining technique to identify interaction patterns between graphical user interface components. Zaidman et al. [16] applied a web-mining technique on program dynamic call graphs, where nodes represent classes and edges represent method invocation. Eisenbarth et al. [9] proposed a formal concept lattice analysis to locate computational units that implement a certain feature of the software system. In contrast to the above techniques, our approach exploits a novel analysis technique to handle large sizes of the execution traces, and allows an intuitive and promising process of feature to component allocation.

We classify approaches to design pattern recovery (focus of this paper) into two major categories, as follows.

Structure-based pattern detection. In this category, the detection process identifies pattern instances that have the same pattern class structure as a target pattern. Nija Shi et al. [14] propose an approach to discover the GoF patterns from Java source code based on data-flow analysis on abstract syntax tree in terms of basic blocks. Lucia et al. [12] propose a two-phase approach to recover structural design patterns, where in the first phase the number of candidate patterns are reduced through analysis of class diagram structure, and in the second phase the real patterns are identified by user inspection.

Matrix-based pattern detection. In this category, the approaches store the inter-class relations in the software system as well as the target design patterns into different matrices. Thus, the pattern matching process is accomplished by matrix matching. Nikolaos et al. [15] present an automatic approach which uses a similarity score algorithm to detect design patterns. The design pattern detection is accomplished by calculating the similarity score between the matrices of system and those of target design patterns.

3. Proposed framework

Figure 1 illustrates the proposed approach for design pattern recovery. The framework consist of *dynamic analysis* to assign system features onto a set of system classes; and *pattern detection* to locate the instances of individual patterns in the software system, by comparing the target patterns in the pattern repository with software's class structure.

Dynamic analysis. The proposed dynamic analysis operates on the run-time execution traces of a set of subject features to locate the corresponding low-level system components that implement each feature. This process consists of the following steps: i) feature-specific scenario set generation; ii) execution traces generation; iii) execution pattern extraction from execution traces; and iv) execution pattern analysis.

Pattern detection. The proposed design pattern detection process consists of two phases *approximate matching* and *structural matching*. In the approximate matching phase, through identifying the eligible candidates for the main-seed class of the target design pattern, we reduce the search space for a target design pattern to a list of source-class clusters, each of which contains a candidate main-seed class. In the structural matching phase, we identify the structurally matched design pattern instances within the list of source-class clusters. The detail description of these two phases are discussed in Sections 4 and 5.

4. Dynamic analysis

We propose a dynamic analysis technique to locate the source code implementation of key features in object-oriented systems, which is an enhancement of the previous

work presented in [13]. In the remaining of this section we will give a description for the process of dynamic analysis.

4.1. Execution pattern extraction

Scenario selection. According to the knowledge about the application domain, available documents, and user's guide of the subject system, we generate a set of relevant task scenarios where all scenarios share a specific software feature. We call this set of scenarios as feature-specific scenario set.

Execution trace generation. In this step, we use Eclipse Test and Performance Tools Platform (TPTP) [2] to instrument and collect execution information from the software system. By running scenarios of the feature-specific scenario set on the instrumented software system, we obtain the execution traces of each scenario in the form of entry/exit listings of the object invocations.

Execution pattern generation. By applying a sequential pattern mining algorithm on the execution traces of the specified feature, we can obtain the execution patterns of the feature. Here we use a modified version of the sequential pattern mining algorithm by Agrawal [4].

4.2. Execution pattern analysis

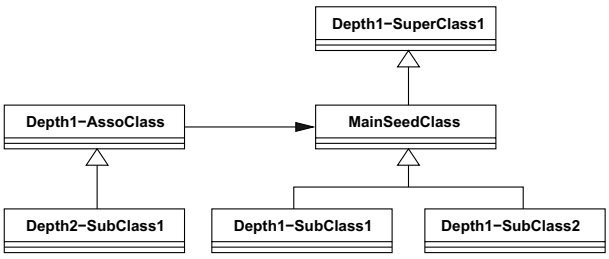
After obtaining the execution patterns of several specific features, we use concept lattice analysis to cluster the group of classes in the execution patterns that exclusively correspond to each specific feature; as well as the class clusters that are common to every scenario set. In this context, the clusters of common classes appear in the upper region of the lattice, and the clusters of feature-specific classes appear at nodes in the lower region of the lattice. Thus, a mapping between the software feature and its implementation is obtained at the bottom of concept lattice.

5. Design pattern detection

To avoid the combinatorial explosion in pattern detection process, we present a two-phase and semi-automated design pattern detection process where each design pattern is populated around a main-seed class.

5.1. Pattern description

Formally, a design pattern p can be represented as a tuple $< \mathcal{C}, \mathcal{R} >$, where $\mathcal{C}$ is a set of *pattern-classes* $\{c_1, ..., c_k\}$ and $\mathcal{R}$ is a set of *inter-class relations* among these pattern-classes. For two pattern-classes c_i and c_j in $\mathcal{C}$, $ShortestPath(c_i, c_j)$ returns the minimum number of inter-class relations traversed from c_i to reach c_j, regardless of the type of the inter-class relations [5]. The *Degree* of a pattern-class c_i in $\mathcal{C}$, denoted as $deg(c_i)$, is the number of the direct inter-class relations between c_i and all the other




```
1  Begin-PDL
2    Pattern : TargetPattern
3      Main-seed class : MainSeedClass
4        Depth1 :
5          Inherit_From :
6            Depth1-SuperClass1
7          Inherited_By :
8            Depth1-SubClass1;
9            Depth1-SubClass2
10       in_Association :
11         Depth1-AssoClass
12     Depth2 :
13       Seed-Depth1: Depth1-AssoClass
14           Inherited_By :
15             Depth2-SubClass1
16     AbstractClasses :
17       Depth1-SuperClass1
18   End-Pattern
19 End-PDL
```

Figure 2. Class diagram and PDL description of a target pattern.

pattern-classes in the design pattern p.

Main-seed class. We observe that for each design pattern presented in [11], there exists at least one pattern-class which can reach any other pattern-classes in the design pattern within a shortest path value 2. We refer to this kind of pattern-class as *potential main-seed class*, whose formal definition is given below.

Potential main-seed class. In a design pattern $p = \langle \mathcal{C}, \mathcal{R} \rangle$, a potential main-seed class, denoted as c^{pms}, is a pattern-class $c^{pms} \in \mathcal{C}$ such that $\forall c_i \in \mathcal{C} \bullet ShortestPath(c^{pms}, c_i) \leq 2$. C^{pms} is referred to the set of all the potential main-seed classes in the design pattern p.

We propose a Pattern Description Language (PDL) to describe a generic pattern. PDL provides a convenient way to describe a design pattern in a precise way and allows the user to define any other pattern they desire to discover. Figure 2 presents the class diagram of a target pattern and its corresponding PDL description.

5.2. Pattern detection

The pattern detection consists of a two-step matching process, as: *approximate matching* to generate a ranked list of eligible candidate instance patterns; and *structural matching* to identify the structurally matched instance patterns within the ranked list of instances.

Approximate matching. In approximate matching, the main goal is to reduce the search space to a number of instance patterns that are sufficiently close to the target pattern. In this context, we specify a set of attributes for the main-seed of the patterns (both target pattern and instance patterns) whose values are used to compare these two patterns. Hence, we can rank eligible instance patterns in the search space and generate a short list of approximately similar instance patterns to the target pattern. The main-seed attributes include the number of *Inherit_From*, *Inherited_By*, *Association* and *Abstract* relations.

As shown in Figure 2, the main-seed class "$MainSeedClass$" possesses one *Inherit_From* relation, one *Association* relation, and two *Inherited_By* relations. Considering a search space as a set of classes $SP = \{c_1, c_2, ..., c_n\}$, for each class $c_i \in SP$ we define an attribute vector $Attr(c_i) = [a_1, ..., a_k]$ with cardinality k. Given the main-seeds c_t of the target pattern and c_i of the instance pattern, the approximate similarity function sim_{apx} is defined as:

$$sim_{apx}(Attr(c_i), Attr(c_t)) =$$

$$\begin{cases} \Delta(Attr(c_i), Attr(c_t)) & Attr(c_i) \geq Attr(c_t) \\ 0 & Else \end{cases}$$

$$\Delta(Attr(c_i), Attr(c_t)) = 1 - \frac{\sum_{j=1}^{k}(Attr_j(c_i) - Attr_j(c_t))}{\sum_{j=1}^{k} Attr_j(c_i)}$$

where $Attr(c_i) \geq Attr(c_t)$ means that the value of each element in the attribute vector $Attr(c_i)$ is greater than or equal to that of attribute vector $Attr(c_t)$. In this context, function sim_{apx} computes the approximate similarity value between the target pattern (represented by the main-seed class c_t) and the candidate instance pattern (represented by main-seed class c_i).

Algorithm "ApproximateMatching" receives the search space, class relation matrices, target pattern, and a cut-off threshold similarity value, and returns the list of eligible candidate instance patterns. The algorithm utilizes the function "$ComputeAttrValue()$" to compute the attribute values of a main-seed using the class relation matrices; and function "$GeneratePattern()$" to compose an instance pattern with two level classes using every class c_i (in different iterations) from the search space.

Systems	Version	# Classes	#Files	#LOC
JHotDraw	5.1	172	144	8419
JHotDraw	6.0b1	405	289	21091
JHotDraw	7.0.7	331	309	32122

Table 1. Statistics for the subject systems.

Structural matching. Structural matching algorithm deals with the identification of all the instances of the target pattern within a candidate instance pattern[1] obtained from the aforementioned approximate matching. The algorithm receives a candidate instance pattern, target pattern, and the class relation matrices. It returns one or more identified instance patterns within the candidate instance pattern. The algorithm utilizes the functions $GetDepth1Classes()$ and $GetDept2Classes()$ to retrieve the corresponding depth1 and depth2 classes from the PDL representation of the target pattern.

After a pattern instance is detected, a further user-assisted verification has to be performed to check whether the detected pattern is actually implemented within the subject software system or not. This verification is performed through browsing the source code and consulting with the existing design documents.

6. Case study

In this section, we discuss the results of applying the proposed approach on a Java open-source project, *JHotDraw* [1]. JHotDraw is a Java GUI framework which is used to draw two-dimensional graphics and it contains many instances of design patterns in its implementation.

Based on discussion in Section 1, we apply our proposed approach on three versions of JHotDraw, ver5.1, ver6.0b1 and ver7.0.7 to extract reusable software artifacts. The experiments are performed on a Windows XP professional edition running on a PC with a 1.5GHZ centrino processor, 512M bytes memory and 1G bytes virtual memory.

Table 1 presents several system statistics from three versions of JHotDraw systems in the case study. Because of space limitation, the results of execution trace extraction and execution pattern mining for 10 features of the three versions of JHotDraw systems are not presented in this paper, however similar experimentation can be found in our previous work [13]. In a further step, we supply the resulting execution patterns obtained from the sequential pattern mining to a concept lattice generation tool, ConExp [3]. Finally, we generate a search space by collecting all classes of the feature-specific concepts and augmenting this space by adding two levels of immediately related classes.

[1]Note that a target pattern usually has fewer classes than the candidate instance pattern, hence it may match with more than one sub-pattern instances within the candidate pattern.

	Rectangle	Round Rectangle	Ellipse	Polygon	Line	Move	Delete	Group	LineConnection	Text
Adapter	0/0/0	1/1/1	0/0/0	2/1/1	1/1/1	0/0/0	0/1/0	1/0/1	2/2/2	0/1/0
Observer	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	1/1/0	0/0/0	2/2/0	0/0/0	0/0/0
Proxy	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0
Decorator	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0
Strategy & State	1/1/1	1/1/1	1/1/1	1/1/0	1/1/0	1/1/1	1/2/0	1/0/1	5/4/3	1/2/1

Legend: A / B / C
A: data of JHotDraw 5.1, B: data of JHotDraw 6.0b1, C: data of JHotDraw 7.0.7

Table 2. Results of mapping between target patterns and 10 features in three versions of JHotDraw.

In the following phase of the experimental study, we apply pattern detection algorithms "ApproximateMatching()" and "StructuralMatching()" (discussed in Section 5) on the search space to detect all the pattern instances of the target patterns in the pattern repository. We describe structural information of each pattern using the proposed pattern definition language (PDL) and store it into the pattern repository. Currently, our pattern repository contains the following patterns: *Adapter, Proxy, Observer, Decorator, Bridge* and *Strategy & State*. To filter out the false-positive patterns in the detected pattern instances, we perform a manual verification on these resulting pattern instances by inspecting the corresponding source code. To correlate a detected pattern instance to a software feature, we check the overlap between the highly related classes of the feature (obtained from concept lattice) with the participating classes of the pattern instance. If there is an overlap, it means that there exists a relation between the feature and the pattern instance. Table 2 presents the correlation of detected pattern instances to the 10 features of the three versions of JHotDraw systems. The value in each entry of the table represents the number of the pattern instances that are correlated with the corresponding feature.

7. Conclusions

In this paper, we presented a two-phase approach to identify individual design patterns within a subject software system as a means to assist the construction of a reference architecture for a family of software systems, or for different versions of the same system. The main advantage of our approach over the existing approaches is incorporating dynamic analysis and feature localization in source code. This allows us to perform a goal-driven design pattern detection and focus ourselves on design patterns that implement specific software functionality as opposed to conducting a general pattern detection which is susceptible to high complexity problem. We have successfully experimented with JHotDraw system which is considered as a benchmark for design pattern recovery.

References

[1] Jhotdraw start page. http://www.jhotdraw.org, 2006.

[2] The eclipse test and performance tools platform, 2006. http://www.eclipse.org/tptp.

[3] Formal concept analysis toolkit version 1.0.1. http://sourceforge.net/projects/conexp.

[4] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE '95: Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14, 1995. IEEE Computer Society.

[5] G. Antoniol, R. Fiutem, and L. Cristoforetti. Design pattern recovery in object-oriented software. In *IWPC '98*, pages 153–160. IEEE Computer Society Press, June 1998.

[6] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. Pulse: a methodology to develop software product lines. In *Proceedings of SSR '99*, pages 122–131, New York, NY, USA, 1999. ACM Press.

[7] J. Bayer, J.-F. Girard, M. Wurthner, J.-M. DeBaud, and M. Apel. Transitioning legacy assets to a product line architecture. In *Proceedings of ESEC/FSE-7*, pages 446–463, London, UK, 1999. Springer-Verlag.

[8] P. Clements and L. Northrop. A framework for software product line practice. Technical report, www.sei.cmu.edu/productlines/framework.html, 2004.

[9] T. Eisenbarth, R. Koschke, and D. Simon. Derivation of feature component maps by means of concept analysis. In *Proceedings of IEEE CSMR'01*, pages 176–179, March 2001.

[10] M. El-Ramly, E. Stroulia, and P. Sorenson. Recovering software requirements from system-user interaction traces. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 447–454, New York, NY, USA, 2002. ACM Press.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[12] A. D. Lucia, V. Deufemia, C. Gravino, and M. Risi. A two phase approach to design pattern recovery. In *in Proceedings of CSMR07*, pages 297–306, Amsterdam, Netherlands, 2007. IEEE CS Press.

[13] H. Safyallah and K. Sartipi. Dynamic analysis of software systems using execution pattern mining. In *ICPC '06: Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 84–88, Athens, Greece, 2006. IEEE Computer Society.

[14] N. Shi and R. A. Olsson. Reverse engineering of design patterns from java source code. In *ASE '06*, pages 123–134, 2006. IEEE Computer Society.

[15] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. Halkidis. Design pattern detection using similarity scoring. In *Software Engineering*, pages 896–909. IEEE Transactions on, Nov. 2006.

[16] A. Zaidman, T. Calders, S. Demeyer, and J. Paredaens. Applying webmining techniques to execution traces to support the program comprehension process. In *CSMR '05: Proceedings of the Ninth European Conference on Software Maintenance and Reengineering (CSMR'05)*, pages 134–142, 2005. IEEE Computer Society.

Active Ontologies – An approach for using Ontologies as Semantic Web Services Interfaces

Tiago Cordeiro Marques[1], Marcio Gurjao Mesquita[1], Julio Cesar Campos Neto[1], Pedro Porfírio Muniz Farias[1]

[1] Universidade de Fortaleza (UNIFOR)
Washington Soares, 1321 J-30
60811-341 Fortaleza-CE. Brasil
55-85-34773268

{tiago.marques.mail, marciomesquita}@gmail.com, {julioccneto, porfirio}@unifor.br

Abstract. In this paper we define the concept of Active Ontologies, i.e. ontologies that function as interfaces indicating a set of web services that must be implemented by the agents that commit to the ontology. In the specific case of ontologies modeled in OWL, it is presented an implementation of the Active Ontologies concept called Active OWL. In Active OWL, classes and properties are annotated by linking them with four types of web services that are called by agents to exchange messages. The semantics of each type of web service is fixed using a KQML performative style, so web services can only be specified to getting, listing, putting and deleting ontology individuals and properties values. The arguments and results of these web services correspond to instances and properties values of the ontology associated with each web service. In order to use ontologies as interfaces, this work proposes an addressing convention to WSDL files that permits to locate them knowing the host and the ontology/class used.

1 Introduction

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network [15] and OWL (Ontology Web Language) [9] plays a central role in the interaction scenerio proposed by the Semantic Web. Ontologies written in OWL provide meaning for data that can be exchanged through the network.

An ontology is an explicit specification of a conceptualization [6]. Following the format proposed by W3C, ontologies represented in OWL, represent classes, attributes, properties and individuals but they do not model behavior, i.e. methods. Web services, on the other hand, do model methods.

According to W3C [16], Web services have interfaces described in a machine-processable format (specifically WSDL [20]). Other systems interact with the Web service in a manner prescribed by its description using SOAP (Simple Object Application Protocol) messages [21], typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

The paradigm of Object Oriented Programming (OOP) recognizes the convenience of encapsulating methods, classes, attributes in the same abstraction. But, although the integration between Web services and OWL seems natural, OWL and the associated Web services specifications, i.e. SOAP and WSDL, are orthogonal.

The main reason for this separation was the independent way the two technologies were conceived. Semantic Web was proposed by W3C while Web services were created by a consortium of companies (Microsoft, IBM and others). Besides that, when the Semantic Web was proposed, the main Web services specifications, WSDL and SOAP, did not exist yet. On the other hand, when WSDL and SOAP were proposed, XML was already a W3C recommendation but RDF and OWL were not.

Several proposals have been made to bring semantics to Web services such as OWL-S [10], SWSF [14], WSMO [18], WSDL-S [17], and SAWSDL [13]. Semantic descriptions allow Web services to be understood and correctly interpreted by machines [4].

A common starting point among these proposals is that they assume that Web services and Ontologies were built separately and then try to provide semantics to Web services by associating them to ontologies. This is a very complicated approach. Making an analogy with Object Oriented Programming, this approach corresponds to specify methods and classes separately and later try to match them using some kind of metadata. Each parameter in a Web service could refer to a different class in a different Ontology. Each Ontology has a different semantics, so it ends up in a big maze when we associate a Web service to several Ontologies in order to give semantics to the Web service.

This article follows the direction used in OOP languages and proposes the concept of Active Ontologies, where ontologies and Web services are specified together. In this approach, ontologies are seen as interfaces. In programming languages, like Java, interfaces are composed of attributes and abstract methods. A class that

implements an interface must follow a contract, i.e. the class must implement the interface's abstract methods.

Therefore, it is defined that an agent implements an ontology if it accepts the meaning of its classes, attributes, individuals, properties, and also implements the Web services associated with this ontology.

The word "Active" means that portions (generally instances) of ontologies will be retrieved dynamically through calls to Web services. The Web services will be built for this specific functionality.

In this article, an architecture implementing the concept of Active Ontologies is proposed. It is called Active OWL, which defines four types of Web services: **get**, **assert**, **retract**, and **list**. These Web services are responsible for handling instances of classes and properties of the ontology.

Initially, to avoid changing the OWL syntax, Active OWL proposes to associate Web services and ontologies through annotations in OWL. Thus, nothing prevents us from incorporating the proposal in the OWL syntax in future versions, for example by introducing a special type of functional property.

This strategy of integrating Web services and Ontologies can be seen as an alternative to extend the expressiveness of ontologies modeled in OWL.

A central point in this proposal is a convention to addressing URLs that addresses the WSDL files. The address of each WSDL file is composed by two parts: a host part and a class part. Knowing the ontology/class and the host we conclude where is the WSDL is located.

The rest of the article is organized as follows: Section 2 describes some proposals to bring semantics to Web services. Section 3 introduces the concept of Active Ontologies. Section 4 presents Active OWL and Section 5 describes the architecture used in the implementation. Section 6 presents the conclusion and future work.

2 Semantic Web Services

Considering the Semantic Web vision, in which each software agent subdivides its tasks, identifies and calls other agents, Web services can be considered as methods from the agent that is being called.

When one tries to give semantics to a Web service, the goal is neither to clarify what the Web service does, nor to formally specify the computation made by the Web service. The goal is to let them be automatically identified and executed by agents that need their results.

Semantic Web services can be defined as Web services whose properties, capabilities, interfaces and effects are encoded in an unambiguous and machine-interpretable form [12].

It is important to observe that manual, non-automatic composition of Web services also have several proposals. BPEL is the one with greater widespread acceptance. The automatic composition supposed by Semantic Web services is not simple, and it certainly is a great leap beyond manual composition.

Today there are five proposals to bring semantics to web services: OWL-S [10], SWSF [14], WSMO [17], WSDL-S [16] and SAWSDL [13]. Table 1 describes some of their characteristics.

Table 1. Characteristics of some proposals to bring semantics to Web services.

Proposals	Characteristics
OWL-S	It is composed of a main ontology, called **service**, and has three sub-ontologies associated with it, called **service profile**, **service model** and **service grounding**.
SWSF	It is composed of an ontology, SWSO, and a language for the axiomatization of the ontology, SWSL.
WSMO	Its goal is to describe all the relevant aspects of a general service that is accessed through a Web service interface.
WSDL-S	Proposes an approach to add semantics to Web services aligned with industry standards (i.e. WSDL).
SAWSDL	Recently it became a W3C recommendation. It is an evolution of the WSDL-S proposal and defines mechanisms to select which semantic annotations can be added to WSDL components. It is already based on WSDL 2.0.

Considerations about the differences between these proposals can be found in [14] [11] [5] [19].

Active OWL proposes to associate Web services and ontologies via annotations in OWL. The proposal differs from SAWSDL and WSDL-S proposals because in these last cases the annotations are inserted in the WSDL.

Moreover, the differences are not only in the way the annotations are inserted. All proposals listed on Table 1 assume that ontologies and Web services were developed separately. Therefore, a matching between the ontologies and the Web services is necessary. The purpose of annotations in Active OWL is to provide this match. In this approach the ontology and its associated Web services are specified together.

If ontologies are conceptualizations where classes, properties, and individuals are specified, the Web services define methods that can be invoked remotely. It seems natural that both tasks should be specified together, in one place, and not isolated, what would demand a complicated match between the specifications later.

3 Active Ontologies

This section starts explaining two underlying motivations for the proposal of Active Ontologies. These motivations are related to the need of distinction between static and dynamic knowledge and the need of distinction between collective and particular knowledge.

Generally, in active ontologies Web services are used to retrieve the dynamic knowledge and the particular knowledge of each agent.

Next, we show that active ontologies could be seen as interfaces in the sense used in OOP. If an agent adopts an active ontology, it also accepts the responsibility for implementing the methods associated to it.

Finally is presented an address convention to WSDL files.

3.1 Shared Knowledge versus Not Shared (Particular) Knowledge

Ontologies are seen as a shared conceptualization which, in fact, is necessary to any message exchange, no matter if it is between people or between systems. Nevertheless, it is necessary to distinguish between this shared knowledge, formalized through the ontologies, which we call Collective Knowledge, and each agent's particular knowledge. This particular knowledge comprises the knowledge that is not shared between agents and it is the content of the majority of messages exchanged between them.

For example, if an agent A asks agent B what is its age, both need to know the meaning of "age". The term "age" must belong to an ontology known to both agents. However, agent A should not know agent B's age until B reveals it. This knowledge about agent B's age is particular only to agent B. The content of the messages exchanged between A and B is composed of particular knowledge of each agent. On the other hand, the collective knowledge is known by both agents a priori, thus the agents don't need to be informed about it through messages exchanged during communication.

Other example can be given from the situation used by Tim Berners Lee to illustrate a scenario using the Semantic Web [3]. In that situation, the agent of a patient's son should query the Semantic Web for available appointment times of several doctors. Each doctor would also have an agent capable of providing such information. Based on this and other information, the first agent should select a doctor to book an appointment.

It is reasonable to consider that all doctors understand the same ontology (or that we could select only the doctors that understand it) and also that concepts such as "Next Available Appointment Time" and "Medical Specialty" should be present in this ontology.

Each doctor should understand the meaning of these concepts (this is a collective knowledge) but if questioned about them, each one would probably give a distinct answer. The "Next Available Appointment Time" and the "Medical Speciality" are particular knowledge to each doctor. If they were present in the ontology, there would be no reasons to query about them.

3.2 Static Knowledge versus Dynamic Knowledge

Besides collective and particular knowledge, there are also two other natures of knowledge: the static knowledge and the dynamic knowledge. Static knowledge has the default behavior of not changing over time. Dynamic knowledge, on the contrary, is expected to change frequently. For example, the name of a company is a static knowledge because, although possible, it is not probable to change over time. But the stocks quotes of this same company on the stock market are a dynamic knowledge, because it is determined by data that change many times during the same day.

Data and instances, which can be inserted and deleted from databases, are dynamic. Comparatively, the database schema, showing which tables exist and how they are related, is static.

The standards for ontologies proposed so far represent static knowledge, but not dynamic knowledge. Ontologies have been used more often to describe concepts and its relationships, what leads us to represent and handle static knowledge. Although we are able to specify individuals in an ontology, there is no specification on how to insert, delete or update new individuals.

We see ontologies in Active OWL as models that represent collective and static knowledge, but they are associated with Web services to return dynamic and particular knowledge from each agent.

3.3 Ontologies as Interfaces

Active Ontologies can be seen as interfaces in OOP. Thus, each agent that respects an Active Ontology gets the responsibility to implement the methods associated to it.

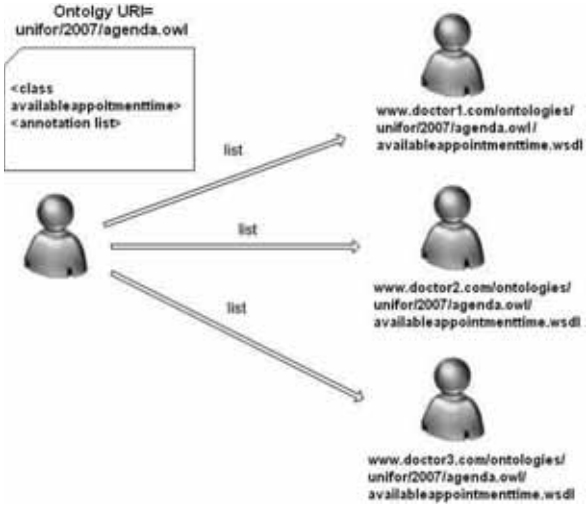


Figure 1. Active Ontologies as Interfaces

On Figure 1 we show an example in which the agent of a patient queries a set of agents that represent doctors and

clinics. Each one of the doctor's agents respects the ontology "unifor/2007/agenda.owl". In this ontology there is an "availableappointmenttime" class with a Web service called "list" linked to it through an annotation.

Thus, each doctor's agent will have a WSDL corresponding to the "availableappointmenttime" class and in this WSDL there must be a Web service called "list".

The patient's agent will be able to call the "list" Web service in each implementation obtaining, in each case, the corresponding list of available appointment times for that specific doctor.

3.4 WSDL Addressing

For each agent, the WSDL file address will be composed of a host part and a class part, according to the grammar shown in Table 2. The host part corresponds to each agent's URL <agent url> whereas the class part is constructed from the ontology's URI <ontology uri> and the class name <class name>. The class part is the same for all the agents that implement the same WSDL.

Table 2. Grammar for the specification of the URL of each WSDL in an Active Ontology.

<wsdl url ::= <host url> <class uri> ".wsdl"
<class uri>::= <"/ontologies/" <ontology uri> "/" <class name>

For example, in Table 3 we show what would be the URL of the WSDL implemented for the last agent in Fig 1.

Table 3. Example URL for the WSDL implemented by an agent.

<host url> = "www.doctor1.com"
<ontology uri> = "unifor/2007/agenda.owl"
<class name> = "availableappointmenttime"
<class uri> = "/ontologies/unifor/2007/Agenda.owl/ availableappointmenttime"
<wsdl url> = "www.doctor1.com/ontologies/Unifor/ 2007/agenda.owl / availableappointmenttime.wsdl"

Using the grammar just shown, the URL for the WSDL can be easily derived from the ontology's URI and the URL of each agent. Alternatively, the WSDL address could be obtained by querying a UDDI server (or other server built specifically for that purpose).

In the case of exists a set of agents that commit to an ontology, the proposed convention permits find each agent´s WSDL file that corresponds to the ontology classes. Also is possible to test if an agent commit to an ontology verifying the presence of the WSDL files in the agent´s host.

4. Active OWL

Active OWL applies to OWL ideas related to Active Ontologies. The approach used to associate Web services to ontologies is to annotate classes and properties indicating the existence of four types of Web services: GET, PUT, DELETE and LIST.

The semantic of annotations are analogous to semantic of KQML performatives ASK, TELL and UNTELL but we choose to use the syntax of HTTP verbs: GET, PUT and DELETE. The annotation LIST was included by the stateless characteristics of web services.

Table 4. Definition of GET, LIST, PUT and DELETE annotations

```
<owl:DatatypeProperty rdf:ID="get">
    <rdf:type rdf:resource="&owl;AnnotationProperty"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="list">
    <rdf:type rdf:resource="&owl;AnnotationProperty"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="put">
    <rdf:type rdf:resource="&owl;AnnotationProperty"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="delete">
    <rdf:type rdf:resource="&owl;AnnotationProperty"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

Annotations are defined in OWL by Annotation Properties. Table 4 indicates the definition of the annotations.

Definitions in Table 4 use the OWL 1.0 syntax, since OWL 1.1 has not been recommended yet. One of the changes proposed for OWL 1.1 is related to annotations. If this change is really confirmed, the solution proposed by Active OWL can be changed with no further problems.

For each annotation in classes and properties, the following Web services must be implemented:

Annotations in classes

- The *get* annotation indicates the existence of a *get(rdf:id)* Web service which receives the individual's identifier by parameter and returns the respective individual.
- The put annotation indicates the existence of a *put(individual)* Web service which receives the individual's corresponding XML by parameter and adds it to the class' individuals set.
- The *delete* annotation indicates the existence of a *deletet(rdf:id)* Web service which receives the individual's identifier by parameter and removes it from the class' individuals set.
- The *list* annotation indicates the existence of a *list()* Web service which lists all the class' individuals.

Annotations in properties

For each class C_i of the set $D(P)=\{C_1, C_2, ..C_n\}$ which constitutes the domain of a property P.

- If P is functional, the *get* annotation indicates the existence of a *getP(rdf:id)* Web service which receives the individual's identifier by parameter

and returns the value of the individual's respective P property.

- The *put* annotation indicates the existence of a *putP (rdf:id, value)* Web service which receives the individual's identifier and the value of the P property by parameter.
- The *retract* annotation indicates the existence of a *deleteP(rdf:id, value)* Web service which receives the individual's identifier and the individual's respective P property value.
- The *list* annotation indicates the existence of a *listP(rdf:id)* Web service which lists all the values of the P property for the individual whose identifier was passed by parameter.

According to the specifications of Table 4, each annotation has a value of type String. Normally, this value will be an empty String. However, this value will be used if it turns out to be necessary to override the values of <class uri>, whose default value is retrieved as pointed in Table 1, or if it is necessary to indicate new parameters for the Web services call.

The syntax of the String which corresponds to the value of each annotation follows the grammar below:

Table 5. Grammar for the values of annotations

<anotation> ::= "ClassUri" = " <class uri> " , parameters = (" <parameter> ["," <parameter>]* ")"
<parameter> ::= String

All mentioned Web services retrieve and update only the individuals that correspond to the particular knowledge of each agent that implements the ontology. That is, they don't retrieve or update the shared knowledge represented in the ontology.

The *get* annotation is defined only for functional properties. With maturing of WSRF specification [8], which allows state management between Web service calls, it will be possible to extend the *get* annotation for non-functional properties.

Annotations cannot be applied to anonymous classes or classes built from expressions. Annotations are not inherited, that is, a child class must specify its own Web services, which will not be inherited from the parent class.

On the current specification, execution errors on Web services should be treated by the application. An expected extension for the current specification would be the treatment of such errors.

Security issues are treated on the level of each WSDL through the appropriate specifications.

5. IMPLEMENTATION ARCHITECTURE

Active OWL was implemented using annotations in functional properties. In the implementation, Jena [7], Axis [1], and AspectJ [2] were used to handle the ontologies, Web services, and aspects, respectively.

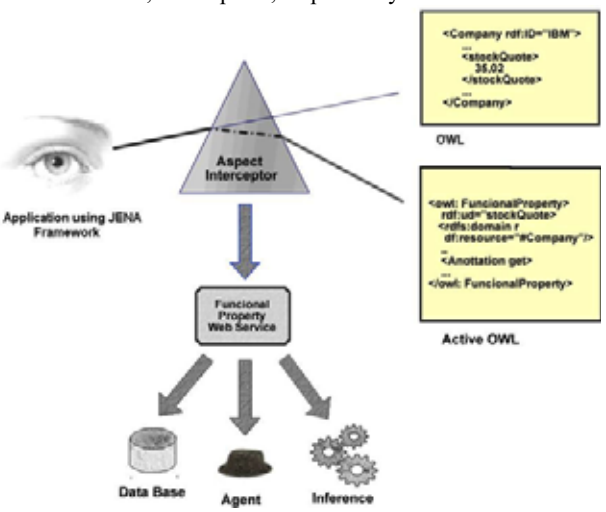


Figure 2. Aspects usage to implement Active OWL

The use of AOP brings the advantage of not being necessary to change any line of code in JENA framework. Some new methods in JENA were introduced in the form of advices in AOP. When annotations are found in functional properties, the advised code calls the respective Web service.

Figure 2 shows the architecture used. The aspect code works like a prism that changes the calls to the ontologies. The ontology is coded in Active OWL syntax, but the program that uses the ontology perceives it as an ontology coded in OWL.

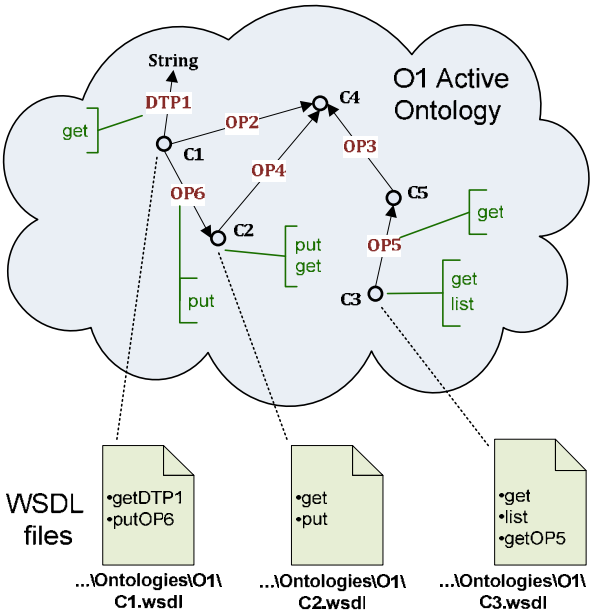


Fig 3. WSDL files used to implement the O1 Active Ontology

In the Figure 3 example we have: the classes C1, C2, C3, C4 and C5; The data type property DTP1; The object properties OP2, OP3, OP4, OP5 and OP6. The domain and the range of the properties are indicated using arrows.

The C1Class is not annotated but the data type property DTP1 is annotated with "get" and the object property OP6 is annotated with "put". The C2 Class is annotated with "put" and "get". The C3 Class is annotated with "get" and "list", also the Object property OP5 is annotated with "get". The C4 and C5 Classes are not annotated. Therefore, to implement the O1 Active Ontology three WSDL files will be created, as indicated in Figure 3.

When GetPropertyValue method from Jena framework is invoked, which is responsible for inspecting the value of a property, the aspect interceptor will intercept the call and invoke the Web service specified in the annotation (i.e. a Functional Property Web service). The result of the Web service call will be shown as if it was taken from the resulting OWL file.

The Web service can compute a function or query for a property value invoking another agent, querying a database or invoking an inference engine to obtain the value.

6. CONCLUSION

This article presents the concept of Active Ontologies and an implementation of the same concept, called Active OWL for the OWL language. Active Ontologies are characterized by the definition of ontologies and the Web services associated with them. Therefore, Ontologies are now seen as interfaces that force those who use them to implement the Web services that are associated with them. Such Web services are responsible for retrieving particular and dynamic knowledge of each agent.

Since Web services and ontologies are now specified together, the complex operations of semantic matching of W3C proposals are not necessary. Web services Composition is made implicitly and transparently as queries are made on ontology individuals.

In Active OWL, four types of Web services (get, put, delete, and list) are associated with ontologies through annotations. Our proposal was implemented only for functional properties, using AspectJ to intercept calls from Jena framework and to invoke associated Web services.

As future work, implementations may add annotations on non-functional properties and classes.

REFERENCES

[1] Apache Axis, http://ws.apache.org/axis/

[2] Aspectj, http://www.eclipse.org/aspectj/

[3] Berners Lee, T., Hendler, J.,Lassila, O.(2001) *The Semantic Web*. Scientific American, May.

[4] Cardoso, Jorge. (2007) *Semantic Web Services: Theory, Tools and Applications* . Information Science Reference, New York.

[5] Fensel, D., Lauser, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J. (2007) *Enabling Semantic Web Services*, Springer Verlag.

[6] Gruber, T.R. (1993) A translation approach to portable ontology specifications. *Knowledge Acquisitions*, 5:199-220.

[7] Jena Semantic Web Framework (2007), http://jena.sourceforge.net/. Accessed July/2007.

[8] OASIS Committee(2006), WS-Resource Framework (WSRF). http://www.globus.org/wsrf/

[9] OWL Technical Committee. (2004) Web Ontology Language (OWL), http://www.w3.org/2004/OWL/. Accessed July /2007.

[10] OWL-S Technical Committee (2004) OWL-S: Semantic Markup for Web Services, W3C Member Submission. http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/. Accessed july 2007.

[11] OWL-S Technical Committee (2004b) OWL-S' Relationship to Selected Other Technologies, W3C Member Submission http://www.w3.org/Submission/2004/SUBM-OWL-S-related-20041122/. Accessed July /2007.

[12] S. McIlraith, T.C. Son & H Zeng.(2001) Semantic Web Services, *IEEE intelligent Systems, Special Issue on the Semantic Web*, 16(2):46-53.

[13] SAWSDL Technical Committee. (2007) Semantic Annotations for WSDL and XML Schema (SAWSDL), http://www.w3.org/2002/ws/sawsdl/. Accessed july /2007.

[14] SWSF Technical Committee.(2005) Semantic Web Services Framework(SWSF), W3C Member Submission. http://www.w3.org/Submission/SWSF/. Accessed July /2007.

[15] W3C Working (2004) Group, Web Services Glossary - Note 11 February 2004 -http://www.w3.org/TR/ws-gloss/.

[16] Web Services Architecture, W3C Working Group Note 11 February 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211. Accessed Oct/07.

[17] WSDL-S Technical Committee. (2005) Web Service Semantics (WSDL-S), W3C Member Submission. Accessed julho/2007 em http://www.w3.org/Submission /WSDL-S/

[18] WSMO Technical Committee (2005) Web Service Modeling Ontology(WSMO), W3C Member Submission. http://www.w3.org/Submission/WSMO/. Accessed July /2007.

[19] WSMO Technical Committee. (2005) Relationship of WSMO to other relevant technologies, http://www.w3.org/Submission/WSMO-related/. Accessed July/2007.

[20] WSDL, Web Services Description Language 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl. Accessed August/2007.

[21] SOAP, Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation 27 April 2007, http://www.w3.org/TR/soap12-part1/. Accessed August/2007.

Failure Prediction Based Self-Healing Approach for Web Service Composition

Yu Dai Lei Yang Bin Zhang Kening Gao
College of Information Science and Technology, Northeastern University
NEU_DaiYu@126.com

Abstract

Web services can be composed together in order to carry out complex transactions or workflows. During the execution of the composite service, if one component service fails, a mechanism is needed to ensure that the failed service can be quickly and efficiently replaced. In this paper, we propose a self-healing approach for web service composition. The key issue of such approach is the failure prediction. Based on prediction of the failure, service which will be failed can be detected as soon as possible. Thus, re-selection process will be started earlier before the invocation of this service. This will make the re-selection process minimize the interrupting time (caused by online re-selection) of the composite service execution and improve the availability of the composite service. The experimentations show better performance of the proposed self-healing approach.

1. Introduction

The QoS of a service may evolve relatively frequently, either because of internal changes or because of workload fluctuations. Thus, a composite service should have self-healing ability. The self-healing ability means that composite service can repair itself if any execution problems occur, in order to successfully complete its execution, while respecting QoS agreements.

Currently, several works focusing on how to establish QoS model and how to do the selection have been studied in Ref. [1, 2]. In these works, the QoS values of component services rely on estimates of service execution parameters. However, at execution time, the actual QoS values will almost surely deviate from the estimates, for example, because of the network load. To avoid this, it is necessary to re-select the composite service. However, as re-selection with global optimization is a NP hard problem, it will need a long time to finish and be likely to interrupt execution of composite service for a long time. Thus, it is needed to minimize the delay caused by re-selection.

With this problem in mind, we present a solution for composite service self-healing. Such approach is supported by a semi-offline re-selection execution environment. In this environment, if the failed service needs not to be invoked, the re-selection process can be done offline without affecting the execution of the composite service. Supported by such environment, in order to enlarge the time of offline re-selection, a failure prediction is proposed. Then, when the QoS is predicted a deviation from the expected QoS, a re-selection process will be triggered before the invocation of the service. Through doing so, the extra delay caused by re-selection will be minimized.

2. Related Works

In order to make the composite service recover from the failure with minimal user intervention and make the recovered composite service meet the end-to-end constraint, researchers proposes the QoS-driven adaptation approach for composite service. Based on the replacement composite service idea, researchers [3, 4] propose approaches of backing up a composite service for each component service. Then, when a component service is incurred a failure, the composite service can be easily switch to a replacement one. In Ref. [3, 4], all the replacement composite services are backed up before the execution of the composite service. Such two approaches do not consider the QoS of services during runtime of the composite service. Thus, the replacement one will not be available sometimes.

One of the researching works that do the re-selection process during the composite service execution is the approach in Ref. [5]. In Ref. [5], the re-selection process will be triggered as soon as the actual QoS deviates from the initial estimates. When the failure is found, the execution of composite service will be stopped until the re-selection process is

finished. Thus, this approach can be only used for runtime-unaware application.

Compared with above works, we introduce a failure prediction and a semi-offline re-selection execution environment. Through doing so, the re-selection process will make the re-selection process minimize the interrupting time of composite service execution.

3. Failure Prediction Based Self-Healing

3.1 Preliminaries

In this section, we introduce some basic concepts that will be used in the remainder of the paper.

Definition 1. QoS of Atomic Service. For an atomic service s (which only contains one operation), the QoS of s can be defined as: QoS(s)=<$Q^t(s)$, $Q^p(s)$>, where:

- $Q^t(s)$ is the response time of s. $Q^t(s)=t_p+R/V_{transmission}$, where t_p is the request processing time; R is the amount of data needed to transmit between s and the execution engine and $V_{transmission}$ is the transmission speed.
- $Q^p(s)$ is the cost of invoking s.

The aim of selection is to maximize the fitness function of the available QoS factors; and meet the constraint specified for some of the factors. Such problem can be formulated as equation (1).

$$\begin{aligned} &\max \sum_{i=1}^{N}\sum_{j\in S_i} F_{ij}x_{ij} \\ &s.t \ \sum_{i=1}^{N}\sum_{j\in S_i} Q_{ij}^t x_{ij} \le Q_c^t \\ &\sum_{j\in S_i} x_{ij}=1,\ x_{ij}\in\{0,1\}\ i=1,...,N, j\in S_i \end{aligned} \quad (1)$$

Where, x_{ij} is set to 1 if atomic service j is selected for service class S_i in the workflow and 0 otherwise. Q_{ij} is the QoS values of service j in class S_i. F_{ij} is a fitness function which can be computed as (2).

$$F_{ij}=w_t*\left(\frac{Q_{ij}^t-u_t}{\sigma_t}\right)+w_p*\left(\frac{Q_{ij}^p-u_p}{\sigma_p}\right) \quad (2)$$

Where w_t and w_p are the weights ($0<=w_t$, $w_p<=1$, $w_t+w_p=1$). σ and μ are the standard deviation and average of the QoS values for all candidate services in a service class.

3.2 Approach Overview

Since that in a composite service, the invocation of one web service will not begin until all the predecessors of this service are finished. Usually, the re-selection process is to re-select the failed service and its successors. Then, the re-selection process will not affect the process of composite service execution, when the failed service needs not to be invoked. Therefore, when it needs not to invoke the failed service, the re-selection process and the execution process of composite service can be done asynchronously. Based on this idea, we introduce a semi-offline re-selection execution environment. In this environment, only when the re-selection process is not finished and the failed service needs to be invoked, the composite service will stop its execution until the re-selection is finished.

In this environment, if a service is perceived to incur a failure earlier before its invocation, there will have longer time to do the re-selection offline. This means that the interrupting time of composite service execution caused by the re-selection will be minimized.

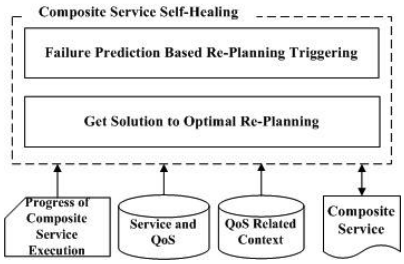


Fig. 1 Framework of The Approach

Based on above basic idea, we propose an approach (Fig. 1) for composite service healing itself. The self-healing contains 2 main parts: failure prediction based re-selection triggering, and getting solution to optimal re-selection problem. We will discuss details of such two problems in the following.

4. Details of the Approach

4.1 Failure Prediction

Compared with changes caused by service providers, changes caused by the network may be occurred more frequently. Changes caused by the network may affect the data transmission speed and thus, affect the response time of composite service. Therefore, in this paper, we will try to predict the data transmission speed. The work of this paper is based on the following assumptions: (a) the failures at different service and communication links are independent; (b) during the data transmission process, data transmission speed is a constant value; (c) price and request processing time of a service is never changed.

We introduce discrete time semi-Markov model [6] for the prediction.

Definition 2. States of Data Transmission Speed. We use th_V_Q to signify the threshold of data transmission speeds in Qualified state.

- If $V(t)>=th\_V_Q$, then $ST(t)$ =Qualified state;
- If $0<V(t)<=th\_V_Q$, then $ST(t)$=Soft Damage state;

- If $V(t)$=0, then $ST(t)$=Hard Damage state.

Definition 5. Semi-Markov Model for Data Transmission Speed. Let Ω be the state space of data transmission speed Ω={1, 2, 3}. $Z=\{Z_t; t>=0\}$ is the random procedure on Ω. If the following conditions are true, we call that $Z=\{Z_t; t>=0\}$ is a semi-Markov process.

- If current state is i, the next state will be entered is j with probability P_{ij}. Especially, P_{ii}=0;
- Given that the next state entered will be j, the time it spends at state i until the transition occurs is a holding time t with distribution $F_{ij}(t)$.

Let $H_i(t)$ be the distribution of holding time in state i, $H_i(t)=\sum_j F_{ij}(t) * P_{ij}$. The average holding time in state i can be signified as μ_i. According to lemmas [6] of semi-Markov model, there exists stationary distribution $\pi=[\pi_1, \pi_2, \pi_3]$ and for each π_j, it can be computed as Eq.(3). Also, let P_i the steady-state occupancy probability of state i, it can be computed as Eq. (4).

$$\pi_j = \sum_{i=1}^{3} \pi_i P_{ij}; \sum_{i=1}^{3} \pi_i = 1 \quad (3)$$

$$P_i = \frac{\pi_i \mu_i}{\sum_j \pi_j \mu_j} \quad (4)$$

In order to predict the future state, it is required to get the context related to data transmission speed.

Definition 3. QoS-Related Context. The QoS-related context observed by observation o can be defined as $QC(o)=<t_{ob}, v, st_{ob}>$, where t_{ob} is observing time; v is the observed data transmission speed at t_{ob}; st_{ob} is state.

The aim of prediction can be described as: if the current state is i, current time is t and the holding time in current state is d, we need to predict the probability of the data transmission speed V_f at future time t_f above the expected speed V_e. Let j be the state V_e belongs to. To solve this problem, we will consider the following two situations:

- State j is same to i

In this situation, the probability can be a sum of the probabilities in the situations with no transition from t to t_f and situation with at least one transition. Then, the probability can be computed as Eq. (5).

$$\begin{aligned} & P((V > V_e) \wedge (D_i > d)) \\ & = P((V > V_e) \wedge (D_i > t_f - t + d | D_i > d)) \\ & + P((V > V_e) \wedge (Z_{t_f} = i) \wedge (d < D_i < d + t_f - t | D_i > d)) \\ & = (1 - F_i(V_e)) * \frac{1 - H_i(t_f - t + d)}{1 - H_i(d)} + (1 - F_i(V_e)) * P_i * \frac{H_i(t_f - t + d) - H_i(d)}{1 - H_i(d)} \end{aligned} \quad (5)$$

- State j is different from i.

If state j is different from i, it means that there exist at least one transition during the duration from t to t_f. Then, the probability can be computed as Eq. (6).

$$\begin{aligned} & P((V > V_e) \wedge (Z_{t_f} = j) \wedge (d < D_i < d + t_f - t | D_i > d)) \\ & = P(V > V_e) * P(Z_{t_f} = j) * P(d < D_i < d + t_f - t | D_i > d) \\ & = (1 - F(V_e)) * P_j * \frac{H_i(t_f - t + d) - H_i(d)}{1 - H_i(d)} \end{aligned} \quad (6)$$

Definition 4. QoS Failure of Service. Considering a service s in composite service CS, if it is predicted that the probability of the data transmission speed during its execution time below the predefined threshold is lower than rd, service s is assumed to incur a QoS failure.

4.2 Re-Selection Triggering

Through failure prediction, the failure of service can be perceived much earlier and the re-selection can begin earlier. Thus, the re-selection will be more likely to finish before the invocation of the failed service and the extra cost caused by re-selection will be minimized. Our approach will trigger the re-selection when a QoS failure is predicted.

4.4 Algorithm for Re-Selection

The re-selection will be done on the re-selected slice of the composite service. The re-selected slice can be generated based on the approach in [5]. Then, the problem of such re-selection can be described as:

$$\begin{aligned} & \max \sum_{i=1}^{N} \sum_{j \in S_i} F_{ij} x_{ij} \\ & s.t \quad \sum_{i=1}^{N} \sum_{j \in S_i} Q_{ij}^t x_{ij} \le Q_c^{t'} \\ & \sum_{j \in S_i} x_{ij} = 1, x_{ij} \in \{0,1\} \; i = 1,...,|FS|, j \in S_i, x_{vu} = 0 \end{aligned} \quad (7)$$

Where, FS is the set of service classes in re-selected slice; $Q_c^{t'}$ is the runtime of original re-selected slice which can be used as runtime constraint for re-selection.

Such problem can be solved by integer programming algorithm [1]. As the limitation of this paper, we will not discuss it in detail.

5. Experimentations

Experimentation 1 is used to test the effectiveness of the proposed semi-Markov model based QoS predicting approach. Simulate test set of data transmission speed according to the Gaussian distribution. The threshold of failure probability is 0.9. The size of QoS-related contexts is 100000. Compare the relation among predicted result, predicting interval and the observation interval between two neighboring contexts. Table 1 gives the result (O_Q is the observation interval between two neighboring QoS-related contexts in QCS; N is the number of predictions; R is the average accurate rate of the predictions.

Table. 1. Semi-Markov Based Predicted Result

	O_Q=0.5s			O_Q=0.1s			O_Q=0.05s		
	I= 10	I= 60	I= 180	I= 10	I= 60	I= 180	I= 10	I= 60	I= 180
N	300	300	300	200	200	200	150	150	150
R%	95	86	80	97	93	83	98	95	92

Table 1 shows that if the observation interval between two neighboring contexts is smaller and the predicting interval is shorter, the prediction will be more accurate. When the observation interval is short enough, although predicting interval is a little bigger, the accurate of prediction will be better also. Thus, through minimizing observation interval, the accuracy of prediction result can be improved.

Experimentation 2 is to test the interrupting time caused by the re-selection process. Randomly generate 10 scenarios with I_Q=0.1s, I=60, k=20, and the threshold of failure probability is 0.9. Compare interrupting time, result of which is shown in Fig 2.

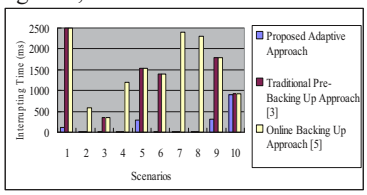


Fig. 2. Comparison of Interrupting Time

Fig 2 shows that the interrupting time of the proposed approach is always the least one among the three approaches. This is because that the when the replacement one is not available and service is failed, re-selection process will start as soon as possible through failure prediction. Thus, the re-selection process will occupy as few as possible execution time of composite service.

Experimentation 3 is used to test the availability of replacement composite service. Randomly generate 10 composite service, for each composite service, simulate 10 failed situations and set I_Q=0.1s, I=60, k=20, the threshold of failure probability is 0.9. Compare success rate of substitution before invoking time of failed service. The result is shown in Fig 3.

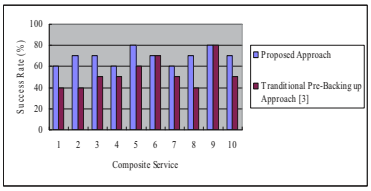


Fig. 3. Comparison of Success Rate

Fig 3 shows that the success rate of the proposed approach is always better than that of traditional pre-backing up approach. This is because that our approach considers the QoS performance of services during the composite service execution, the availability of replacement composite service can be preserved and thus, the success rate will be high.

6 Conclusions

In order to solve the problem of making composite service adapt to dynamic property of services, we propose a self-healing approach for web service composition. The experimentations show better performance of the proposed self-healing approach. In the future work, the prediction approach will be studied more and the proposed self-healing approach will be put into practical applications of service composition.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under Grant No.60773218 and Natural Science Foundation of Liaoning Province under Grant No. 20072031.

References

[1] L. Z. Zeng, B. BENATALLAH, "QoS-Aware Middleware for Web Services Composition", *IEEE Transactions on Software Engineering*, 2004, 30(5), pp. 311-327.

[2] T. Yu, Y. Zhang, and K. J. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints", *ACM Transactions on the Web*, 2007, 1(1), Article 6.

[3] T. Yu, K. J., "Adaptive algorithms for Finding Replacement Services in Autonomic Distributed Business Processes", *In: the 7th International Symposium on Autonomous Decentralized Systems*, Chengdu, China, 2005, pp. 427-434.

[4] C.Girish, D. Koustuv, K. Arun, M. Sumit, and S. Biplav, "Adaptation in Web Service Composition and Execution", *In: IEEE International Conference on Web Services*, USA, 2006, pp. 549-557.

[5] G. Canfora, M.D. Penta, R. Esposito, and M. L. Villani, "QoS-Aware Replanning of Composite Web Services", In: IEEE International Conference on Web Services, USA, 2005, pp.121-129.

[6] M. Malhotra, A. Reibman, "Selecting and Implementing Phase Approximations for Semi-Markov Models", *Communication Statistics-Stochastic Models*, 1994, 9(4), pp. 473-506.

A WEB-BASED DATA MANAGEMENT AND ANALYSIS SYSTEM FOR CO_2 CAPTURE

Yuxiang Wu, Christine W. Chan*
Energy Informatics Laboratory, Faculty of Engineering, University of Regina
Regina, Saskatchewan, Canada
Email: Christine.Chan@uregina.ca

Abstract

Carbon dioxide (CO_2) capture technologies are important for helping to cut CO_2 emissions into the atmosphere, which is now an urgent objective in a world faced with increasing hazards of global warming. Chemical absorption has become one of the dominant CO_2 capture technologies because of its efficiency and low cost. The chemical absorption process involves over a hundred components, which generate a vast amount of data. While it is important to monitor the generated data in order to ensure normal process operations, the monitoring task is complex and automated support is highly desirable. The Data Analysis Decision Support System presented in [2] supports automated monitoring of the CO_2 capture process but suffers from a number of limitations. Its weaknesses include difficulty in data sharing, limited accessibility in a small LAN environment, and inflexibility in data format. The objective of this work is to extend the DADSS into a web-based system that does not have the above limitations. This paper presents the web-based data management and analysis system for carbon dioxide capture process called CO_2DMA. The process of developing CO_2DMA involves software engineering technologies such as the technologies of object-linking and embedding (OLE) for process control (OPC) and web-application development frameworks.

Keywords – web-based; decision support system; carbon dioxide capture; data filtering.

1. Introduction

Fossil fuel is presently the world's most abundant, economical and reliable fuel for energy production. However, the industry now faces a major challenge because the production of fossil fuels, which include coal, crude oil and gas, and the processes currently used for energy production from such fuels, can have adverse environmental consequences. Hence, along with the positive economic advantages of energy production using fossil fuels comes to the responsibility of avoiding their potential misuses and the consequent adverse environmental and climate-change impacts [2].

Carbon capture and storage (CCS) is an approach for cutting the carbon dioxide (CO_2) emissions to the environment by capturing and storing the CO_2 gas. Among various CO_2 capture technologies, chemical absorption of CO_2 is one of the most mature technologies because of its efficiency and low cost.

The highly complex CO_2 absorption process generates a vast amount of data, which need to be monitored, preferably by an automated system. But industry process control systems do not typically provide intelligent data preprocessing or data analysis functionalities. Therefore, it is necessary to construct an intelligent data management and analysis system. The Data Analysis Decision Support System (DADSS) for CO_2 capture process reported in [2] is a step towards filling this need. However, the DADSS is a standalone PC-based system with limited flexibility and connectivity. In this paper we present a web-based CO_2 data management and analysis system (CO_2DMA).

The system presented in this paper, as well as the first prototype discussed in [2], are built based on data acquired from the Pilot Plant CO_2 capture process at the International Test Centre for CO_2 capture (ITC), located at the University of Regina. The CO_2 capture process in the ITC is monitored and controlled via the DeltaV system (Trademark of Emerson Process Management, U.S.A), which is based on the technology of Object-Linking and Embedding (OLE) for Process Control (OPC). OPC standards are widely used in industry process control and manufacturing automation applications [4]. More detailed information about OPC will be provided later.

The paper is organized as follows: Section 2 briefly describes the first prototype of DADSS. Section 3 discusses software engineering techniques used in system development. Section 4 presents some sample test runs of the web-based system. Section 5 concludes the paper.

*Author to whom all correspondence should be addressed.

2. The DADSS System

Briefly, the system described in [2] is a decision support system for pre-filtering and analysis of data captured from the CO_2 capture process. It is called the Decision Support System for analysis of CO_2 capture process, or Data Analysis Decision Support System (DADSS) for CO_2 capture process.

The basic structure of the DADSS is shown in Fig. 1. The controller module of DADSS accepts inputs from the user and activates the model and view modules to perform actions based on those inputs. In effect, the controller is responsible for mapping end-user actions to application responses. The Data Access Object (DAO) module provides a common interface between application and the data storage, such as a database. The DAO module is a way for separating object persistence and data access logic from any particular mechanism or API.

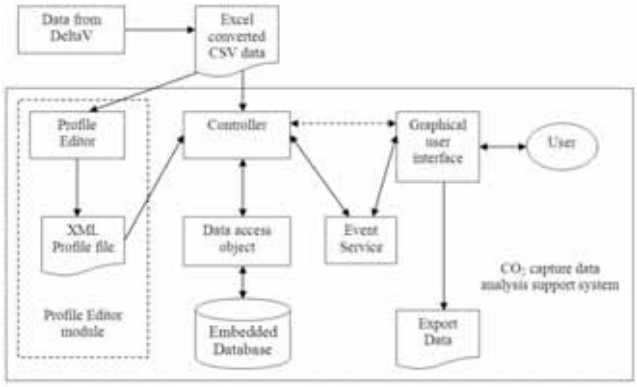


Fig. 1. Structure of Data Analysis Decision Support System for CO_2 capture process

Some limitations that exist in the old system due to its pure client-based features include:

- The system relies on CSV file as its data source. This is not flexible enough for future data analysis.
- Knowledge and data sharing through specific CSV file will be difficult.
- Data access is limited to small LAN environment, and to PC platform.

In order to overcome these limitations, a new web-based CO_2 data management and analysis system is built.

3. CO_2DMA System Development

This section presents the structure of the CO_2DMA, and several software engineering technologies that were used during development.

3.1 System Structure

Briefly, the system consists of four main modules (Fig. 2): (1) OPC Historical Data Access (HAD) Server module, (2) OPC Data Transporter module, (3) Database Server module, and (4) Web Server module. OPC HAD Server usually resides in the same machine with the process control system, which refers to the DeltaV system in ITC. It is the repository where process data are stored, and which can be only accessed by programs with built in HDA standards. OPC Data transporter is a C# (Microsoft® software) program that runs along with the OPC HDA Server in the background. It continually reads data from OPC HDA Server and converts the data into appropriate types in order to transfer them into the Database Server. The Web Server component of the system is responsible for communicating with clients through the internet. Clients send request to and retrieve data from the Web Server. Both communication and data transfer are based on HyperText Markup Language (HTML).

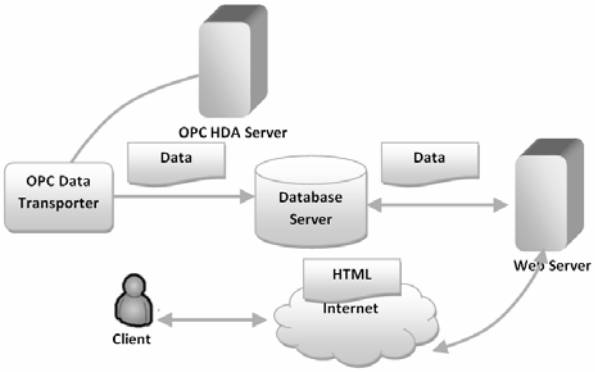


Fig. 2. System Structure

3.2 OPC and OPC Transporter

OPC, which stands for Object-Linking and Embedding (OLE) for Process Control, is basically a series of standard specifications [8]. The OPC standard specifications support communication of real-time plant data between control devices from different manufacturers [9]; the OPC Foundation maintains the standards. Since the foundation was created, more standards have been added.

The purpose of using OPC was to bridge Windows (Microsoft® software) based applications with process control hardware and software applications because these open standards support a consistent method of accessing field data from plant floor devices. OPC servers can provide a method for many different software packages to access data from control devices by defining a common interface.

Despite its advantages, two main drawbacks were found within the OPC technology during interviews with the operators of the CO_2 capture process:

(1) The OPC technology including OPC servers and client applications, are developed based on the windows platform. This presents a problem when data and knowledge sharing needs to be done with an application that is developed on a non-windows platform. Hence, interoperability is not supported.

(2) Only applications that support OPC protocols can access the data in OPC HDA Server, which is where the DeltaV data are stored. As a result, data manipulation and analysis relies on OPC client applications. The data cannot be easily reused by other mature computational tools that do not have OPC interfaces built in.

To address these limitations, the decision was made to build a generic database, which would retrieve data from the real-time control system and store them. We believe the generic database can render our system more flexible and the data reusable. The component called OPC Data Transporter is constructed for accessing, converting and sending data from the OPC HDA Server to our generic database. The OPC Data Transporter is an OPC client application written in C# (Microsoft® software) by using the Historical Data Access (HDA) common library. Currently the transporter runs as a background program within the same machine as the DeltaV control system. It can also reside on a remote machine which physically connects to the control system. In either case, data will be periodically captured from the OPC HDA Server and converted to the correct data type, then stored in the generic database. This approach allows us to protect the control system by isolating it from outside interference, while enabling sharing of data and other useful information.

3.3 Web Server Development

The Web Server plays a key role in our system because it acts as an intermediary between the database component and the user on the internet. The server was constructed using the LAMP software bundle, which includes:

- Linux, a Unix-like computer operating system.
- Apache, an open source HTTP Server.
- MySQL (Trademark of MySQL AB), multi-user SQL database management system (DBMS).
- PHP (Hypertext Preprocessor), a computer scripting language originally designed for producing dynamic web pages.

This LAMP bundle has become widely popular since its inception by Michael Kunze in 1998 because this group of free software could provide a viable alternative to commercial packages [10]. Therefore, the LAMP bundle has been adopted for developing our Web server.

Usually the most time consuming part of building a web server is to program the entire site including design of the user interface as well as construction of the background logical layer. This process was often conducted in ad hoc manner, based neither on a systematic approach, nor quality control and assurance procedures. Recently, different types of web application frameworks supporting different languages have been built. A web application frame work is a software framework that is designed for supporting the development of dynamic websites, web applications and services; the framework is intended to simplify the overhead associated with common activity procedures in web development. The general framework usually provides libraries for database access, template frameworks, session management and code reuse.

In our development of the web server, CakePHP (trademark of Cake Software Foundation) was adopted as the basic framework because of its detailed documentation and ease of use. Based on CakePHP, the system structure of the web server was designed and developed as shown in Fig. 3.

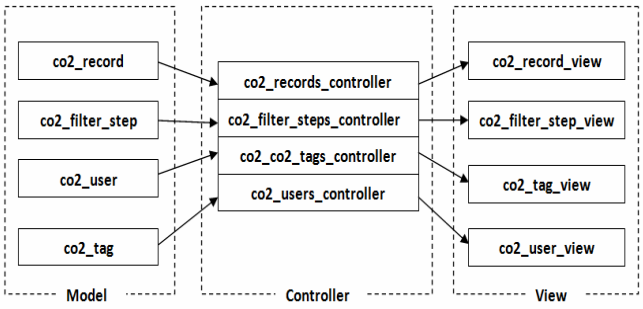


Fig. 3. Web Server Structure

As shown in Fig.3, the structure of our system follows the Model-View-Controller (MVC) architectural pattern used in software engineering. Recently the model has become widely used in web application development.

In the web server system, the model represents a particular database table, and its relationships to other tables and records. The Model also consists of data validation rules, which are applied when the model data are inserted or updated. The View represents view files, which are regular HTML files embedded with PHP code. This provides users with the web page display. The controller handles requests from the server. It takes user input which includes the URL and POST data, applies business logic, uses Models to read and write data to and from databases and other sources, and finally, sends output data to the appropriate view file [12]. This system structure has the advantages of (1) modularizing the code and making it more reusable, maintainable, and generally better; and (2) encapsulating knowledge structure and translating the knowledge into procedures and methods using an object-oriented representation.

4. Sample Run of System

Data on operation of the CO_2 capture process from 4/3/2006 to 4/13/2006 provided by ITC were used to test the CO_2DMA in this sample run.

The difference between the unfiltered data and the data that have been filtered using CO_2DMA can be revealed by examining the two sets of data on the sample variables of 'Heat Duty' and 'FI700', which were selected from the 145 tags. Two trend lines that approximate the data are drawn as shown in Fig. 4 and Fig. 5. The points in the plot of the unfiltered data in Fig. 4 are more scattered because of the high volume of noisy data. After filtering by our system, more than 60 rows of noisy data were filtered out from the 590 rows, and the data points are more clustered together as shown in Fig. 5.

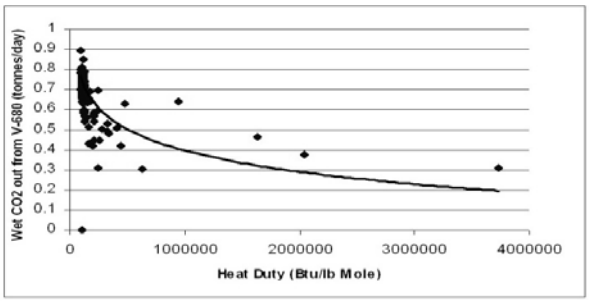


Fig. 4. Plot of data before filtering

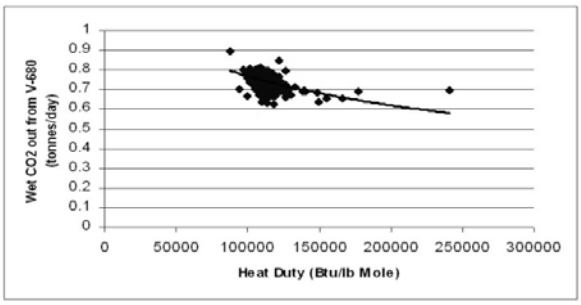


Fig. 5. Plot of data after filtering

5. Conclusion and Future Works

A web-based data management and analysis system for the CO_2 capture process has been developed to overcome the limitations of the existing DADSS. Since the system is built as a web service application, there is no need to install any software in the user's computer. By automatically filtering and processing hundreds of fields of raw data, the CO_2DMA frees users from having to perform data filtering manually; hence, it improves efficiency of the data filtering process.

Currently we are working on enhancing system efficiency by saving the user's preferred filtering procedures in a historical configuration file. In the future, we plan to add curve fitting and graphing functions to the system so that the filtered data can be processed for visual displays inside the system instead of being exported to Microsoft Excel for further charting. Automation of the data filtering step is only the first step in our research agenda. Future objectives include utilizing the data for prediction, planning and control of the CO_2 capture process using artificial intelligence techniques.

Acknowledgement

The authors are grateful for the generous support of grants from Canada Research Chair Program and Natural Science and Engineering Research Council of Canada. We would also like to acknowledge the help and contributions of Robert Harrison, Chuansan Luo, Dr. Paitoon Tontiwachwuthikul, Don Gelowitz and Dr. Raphael Idem.

Reference

[1] IPCC special report on Carbon Dioxide Capture and Storage. Prepared by working group III of the. Metz, B., O.Davidson, H. C. de Coninck, M. Loos, and L.A. Meyer (eds.). Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA, 442 pp.

[2] Robert Harrison, Yuxiang Wu, Hanh Nguyen, Xiongmin Li, Don Gelowitz, Christine W. Chan, Paitoon Tontiwachwuthikul, A Decision Support System for Filtering and Analysis of Carbon Dioxide Capture Data, CCECE 2007. Canadian Conference on Volume, Issue, 22-26 April 2007 Page(s):1380 – 1383.

[3] Riemer, P. (1996). Greenhouse gas mitigation technologies- an overview of the CO_2 capture, storage and future activities of the IEA greenhouse gas R&D programme. Energy Conversion and Management, 37, 665-670.

[4] OLE for process control, http://en.wikipedia.org/wiki/OLE_for_process_control (Mar. 2, 2008).

[5] Knowledge Acquisition, http://www.centc251.org/Ginfo/Glossary/tcglosk.htm (15 Dec 2006).

[6] Knowledge Acquisition, http://www.epistemics.co.uk/Notes/63-0-0.htm (15 Dec 2007).

[7] Durkin, J. Expert system design and development. Englewood Cliffs, New Jersey: Prentice Hall. 1994.

[8] OPC Foundation, http://www.opcfoundation.org/ (11 Mar 2008).

[9] OPC Introduction, http://en.wikipedia.org/wiki/OLE_for_process_control (11 Mar 2008).

[10] Michael Kunze, c't 12/98, page 230 - Freeware Web Publishing System, 1998.

[11] San Murugesan, Yogesh Deshpande, Web Engineering: Managing Diversity and Complexity of Web Application, 2001.

[12] CakePHP Manual, http://manual.cakephp.org/chapter/basic_concepts (11 Mar 2008).

Integrating Random Testing with Constraints for Improved Efficiency and Diversity

Yoonsik Cheon, Antonio Cortes, Martine Ceberio
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968

Gary T. Leavens
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816

Abstract

Random testing can be fully automated, eliminates subjectiveness in constructing test data, and increases the diversity of test data. However, randomly generated tests may not satisfy program's assumptions such as method preconditions. While constraint solving can satisfy such assumptions, it does not necessarily generate diverse tests and is hard to apply to large programs.

We blend these techniques by extending random testing with constraint solving, improving the efficiency of generating valid test data while preserving diversity. For domains such as objects, we generate input values randomly; however, for values of finite domains such as integers, we represent test data generation as a constraint satisfaction problem by solving constraints extracted from the precondition of the method under test. We also increase the diversity of constraint-based solutions by incorporating randomness into the solver's enumeration process. In our experimental evaluation we observed an average improvement of 80 times without decreasing test data diversity, measured in terms of the time needed to generate a given number of valid test cases.

1 Introduction

A random approach to generating test data has the potential for finding faults that are difficult to find in other ways, because it eliminates subjectiveness in constructing test data and increases the diversity of input values. It also facilitates test automation. Our recent work explored random test data generation to unit testing of Java classes annotated with assertions [6]. A test case for a method is constructed dynamically to ensure that it satisfies the precondition of the method under test. If a test case does not satisfy the precondition, it is inadequate to test the method because the precondition is the client's obligation [5].

However, randomly generated tests may not satisfy the program's assumptions. In our case these assumptions are method preconditions formally written in JML [11], an interface specification language for Java. If the preconditions are not trivial, the chances are very low that randomly-generated test data will satisfy them. In our recent experiment we observed that up to 99% of randomly-generated test cases did not meet the preconditions of the methods under test [6].

In this paper we propose an extension to pure random testing to improve the efficiency of generating a given number of valid test cases that satisfy the precondition of the method under test. The key idea of our extension is to integrate constraint solving with random test data generation. For method parameters of continuous or infinite domains such as objects, we generate test values randomly. However, for parameters of discrete and finite domains such as integers, we represent test data generation as a constraint solving problem, where constraints are the assertions of the method precondition that involve the parameters. This extension is based on our observation that about 10% to 50% of methods have formal parameters of discrete and finite domains and the precondition assertions on these parameters can be efficiently solved by finite-domain constraint solvers.

We evaluated the effectiveness of our approach by implementing a prototype tool based on our own random testing tool called JET [6] and an open-source constraint solver called Cream [14]. In our experiments we observed an average improvement of 80 times over pure random testing measured in the time needed to generate a given number of valid test cases that satisfy a method's precondition (see Section 5).

2 Background

Our long term goal is to fully automate unit testing of Java classes, from test data generation to test execution and test outcome decision. The class under test is assumed to be annotated with a JML specification (see Section 3); formal specifications such as method postconditions are used as test oracles. Each method of the class is tested separately, and thus a *test case* consists of a receiver object and argument values. We generate test cases automatically—the subject of this paper—

and perform test executions by invoking the method under test with the generated test data. We use JML's runtime assertion checker to recognize invalid test cases as well as to decide test outcomes; i.e., we interpret certain types of assertion violations, such as postcondition violations, as test failures [5, 13].

Previous work has either generated test data randomly (e.g., [6, 7, 8, 12]) or has generated tests purely by solving constraints (e.g., [1, 3]). In *random testing*, a random object of a class C is obtained via a *call sequence*, consisting of one constructor and zero or more method invocations, such as $C\ o$=**new** $C_0(); o.m_1(); o.m_2(); \ldots; o.m_n()$. In such a call sequence, m_1 through m_n mutate the state of o. Methods m_i and their arguments are selected randomly from appropriate methods of C.

In *constraint-based testing*, test cases are generated by solving constraint satisfaction problems. A constraint satisfaction problem consists of a finite set of variables and a set of constraints on those variables. Each variable is associated with a set of possible values, known as its domain. A constraint is simply a relation on some subset of these variables. A solution to a constraint satisfaction problem is an assignment of a value to each variable from its domain, such that all the constraints are satisfied. There are efficient constraint solvers for finite domains. For example, Cream [14] is a Java class library for solving constraints on finite domains. In Cream, the collection of variables, domains, and constraints are called a *constraint network*. Cream provides several built-in strategies, called *solvers*, that enumerate solutions for constraint networks (see Section 3).

3 Illustration

To illustrate our approach, let us consider the `Account` class given in Figure 1. This class is annotated with JML assertions written as special comments. The keyword **`spec_public`** states that the private field `bal` is treated as public for specification purpose; e.g., it can be used in the specifications of public methods. A method specification precedes the declaration of the method and specifies its precondition (**`requires`** clause), its frame condition (**`assignable`** clause), and its postcondition (**`ensures`** clause). The keyword **`\old`** denotes the pre-state value of its expression and is used in the specification of a mutation method such as the `transfer` method that changes the state of an object.

Consider the `transfer` method and the likelihood of randomly generating a valid test case. To test this method, we need a test case consisting of two `Account` objects—one for the receiver and the other for the argument—and an integer. Let p be the probability of generating an `Account` object successfully, i.e., the probability that all the calls in its call sequence terminate normally. Then, the probability of generating a valid test case is p^2q, where q is the probability that the test case satisfies the method precondition,

```
public class Account {
  private /*@ spec_public @*/ int bal;
  //@ public invariant bal >= 0;

  /*@ requires amt >= 0;
    @ assignable bal;
    @ ensures bal == amt; @*/
  public Account(int amt) {
    bal = amt;
  }

  /*@ requires amt > 0 && amt <= acc.bal;
    @ assignable bal, acc.bal;
    @ ensures bal == \old(bal) + amt
    @   && acc.bal == \old(acc.bal - amt); @*/
  public void transfer(int amt, Account acc) {
    acc.withdraw(amt);
    deposit(amt);
  }

  // The rest of the definition including:
  // Account(Account), deposit(int),
  // withdraw(int), and int balance().
}
```

Figure 1. JML-annotated class

`amt > 0 && amt <= acc.bal`. In a purely random approach, q is 0.25 if we conservatively estimate the probability of satisfying each conjunct to be 0.5.

However, we can be clever by selecting an `amt` value such that it automatically satisfies the precondition, thus improving q to 1. To do this, we solve the constraints on `amt` imposed by the precondition; i.e. we represent the problem of test case generation partly as a constraint satisfaction problem. For this particular case, we need to solve the constraints: $x > 0$ and $x \leq B$, where B is `acc.bal`, the balance of the randomly-generated `Account` object. These constraints can be easily translated to the following Cream code.

```
Network net = new Network();
IntVariable x = new IntVariable(net);
x.gt(0);
x.le(acc.bal);
Solver solver = new DefaultSolver(net);
Solution solution = solver.findFirst();
int valX = solution.getIntValue(x);
```

In Cream, constraints on variables are expressed using framework methods such as `gt` and `le`. The Cream framework also provides a set of arithmetic methods (e.g., `add` and `multiply`) for writing arithmetic expressions.

Our approach improves the probability of generating valid test cases dramatically. Recall that an object in a test case is represented as a call sequence. Thus a test case is set of such call sequences. We can apply constraint solving to each of the method invocations in the call sequence. For example, doing this for the `transfer` method improves not only q but also p—the probability of generating a valid `Account` object—even more dramatically, which has a greater impact on the overall probability.

4 Our Approach

The problem is to generate test cases that satisfy the precondition of the method under test. The key idea of our approach is to solve constraints for values of finite domains such as integer while generating random values for other types such as objects. There are two main issues. The first is how to identify and extract constraints from method preconditions written in JML; not all precondition assertions are constraints on the parameters of interest. The second issue is how to translate the extracted constraints to constraint solving code.

We address the first issue by first desugaring the executable subset of JML precondition assertions to a single boolean expression and then converting it to a *disjunctive normal form*. As in the predicate-based approach to test data generation [16], we consider each disjunct of the disjunctive normal form as a constraint to solve. For the second issue, we define a translation from disjuncts of the normal form to Cream code.

As in [6], the receiver object of a test case is generated randomly, but the arguments are generated in a combination of a random approach and a constraint satisfaction problem. For this, we classify formal parameters of a method into two categories.

Definition 1 *A formal parameter of a method is a* constrained variable *if its declared type is an integral type such as* `int`*. A formal parameter that is not a constrained variable is called an* unconstrained variable*.*

As outlined below, we first prepare the preconditions for possible constraint solving, and then generate test cases. This preparation involves the following steps.

1. Desugar the method precondition to a single boolean expression (see Section 4.2).
2. Convert the boolean expression to a disjunctive normal form (see Section 4.3).
3. For each disjunct of the normal form that has constraints on any of the constrained variables, translate it to constraint solving code (see Section 4.4).

Test case generation is then done by repeating the following until a fixed number of valid tests are generated. We generate random values for the receiver and all arguments as in our previous work [6]. If the generated values do not satisfy the precondition of the method under test, we then find new values for the constrained variables by solving constraints extracted from the precondition.

1. Generate random values for the receiver and all arguments (see Section 4.1).
2. If the values do not satisfy the precondition, find new values for the constrained variables by invoking the constraint solving code (from Step 3 above).

If no constraints were identified for the constrained variables (see Section 4.4) or no solution found, then we repeat the whole process some fixed number of times.

4.1 Random Value Generation

We use the method of [6] to generate the initial random values for the receiver and arguments; e.g., a random object is constructed as a sequence of mutation method invocations preceded by a constructor invocation. However, one important difference is that the receiver and arguments of each method invocation in the object sequence are also generated by using the new approach, because they also have to satisfy the precondition of the invoked method.

4.2 Precondition Desugaring

JML features a great deal of syntactic sugar to enhance Java expression syntax by introducing a rich set of JML-specific expressions and several specification clauses. We desugar the executable subset of a method precondition to a single boolean expression.[1] The desugaring process consists of two steps: desugaring of method specifications and simplification of boolean connectives.

4.3 Disjunctive Normal Form

We use a disjunctive normal form to identify the set of constraints that can be solved independently to find values for the constrained variables that satisfy the precondition of the method under test. A *disjunctive normal form* (*DNF*) is a standardization or normalization of a logical formula which is a disjunction of conjunctive clauses, e.g., $c_1 \vee \ldots \vee c_n$, where each c_i is of the form $e_1 \wedge \ldots \wedge e_m$.

4.4 Constraint Identification

From a method precondition converted to a DNF, we identify constraints on the constrained variables. We assume all Java/JML boolean connectives are already desugared except for conjunction, disjunction, and negation.

Definition 2 *A constrained variable is* executable *in an expression, e, if it has a free occurrence in e other than in a subexpression of a receiver or an argument to a method or constructor call. A boolean-valued expression that contains no logical connectives is an* executable constraint *if it contains at least one executable constrained variable.*

We often use the term "constraint" as shorthand for "executable constraint." The following gives an equivalent characterization of executable constraints.

[1] The executable subset is JML expressions and assertions that are translated to runtime assertion checking code by the JML compiler (`jmlc`) [4].

Theorem 1 *A boolean-valued expression e that contains no logical connectives is an executable constraint if and only if it is of the form $e_1 \diamond e_2$, where both e_1 and e_2 are of an integral type, $\diamond$ is a relational or equality operator, and either e_1 or e_2 contains an executable constrained variable.*

This follows from our definitions of constrained variables (being of integral types) and constraints (being boolean expressions).

Uses of constrained variables are not executable when they occur as arguments to method calls, because Cream does not understand arbitrary methods, and hence it cannot solve for such occurrences. For example, `Math.abs(x) > 10` is not an executable constraint because Cream cannot handle the call to `abs`. (To make it executable, one has to translate the expression manually to one that can be handled by Cream.) On the other hand, in the expression `Math.abs(x - 10) > y`, although `x` is not executable, `y` is, and thus the entire expression is an executable constraint. Cream can thus try to satisfy this assertion when `x` has a random value, even though it does not control `x`'s value.

Definition 3 *A conjunctive clause of the form, $e_1 \wedge \ldots \wedge e_n$, where e_i's do not use disjunction, is an* executable constraint *if at least one e_i is an executable constraint.*

A conjunctive clause that is not an executable constraint may contain a constrained variable, but not one that is executable. If each e_i of the clause is an executable constraint, the solutions of the whole constraint are in general valid test data; otherwise the validity of the test data depends on the e_i's that are not constraints.

4.5 Constraint Solving Code

Given a DNF $c_1 \vee \cdots \vee c_n$, we consider each conjunct clause c_i independently. If c_i is a constraint, we translate it into Cream constraint solving code. The translated Cream code has the following general structure, where x_i's are the constrained variables appearing in the constraint c_i and y_i's are fresh variables to store a solution.

```
Network net = new Network();
IntVariable x1 = new IntVariable(net);
...
IntVariable xm = new IntVariable(net);
⟨Translated constraints of ci⟩
Solver solver = new DefaultSolver(net);
Solution solution = solver.findFirst();
int y1 = solution.getIntValue(x1);
...
int ym = solution.getIntValue(xm);
```

This skeletal Cream code has three parts. It first creates a new constraint network and adds the constrained variables of c_i to the network. It then specify the constraints of c_i using the added constrained variables (see Section 4.6 below). It finally solves the constraints and retrieves a solution.

JML Expression		*Cream Constraint Code*
$x \geq 10$	x	`IntVariable v1 = null;` `v1 = x;`

	10	`int i1 = 0;`
		`i1 = 10;`
		`IntVariable v2 = new IntVariable(net);`
		`v2.equals(i1);`

	$\geq$	`v1.ge(v2);`

$x + y$ < size()	$x + y$	`IntVariable v3 = null;`
		`IntVariable v5 = null;`
		`v5 = x;`
		`IntVariable v6 = null;`
		`v6 = y;`
		`v3 = v5.add(v6);`

	size()	`int i2 = 0;`
		`i2 = size();`
		`IntVariable v4 = new IntVariable(net);`
		`v4.equals(i2);`

	$<$	`v3.lt(v4);`

Figure 2. Sample translation of a conjunct $x \geq 10 \ \wedge \ x + y < \text{size}()$, where x and y are constrained variables.

4.6 Constraint Translation

Given a conjunctive clause of the form $e_1 \wedge \ldots \wedge e_n$, we translate each e_i into Cream if it is a constraint; otherwise, we ignore it because it does not constrain the parameters of interest or the constraint cannot be handled in Cream. For this, we defined a set of translation rules and the rules systematically translate JML expressions to Cream code by converting Java/JML operators to Cream framework methods and by introducing temporary variables as necessary. As an example, consider a conjunctive clause $x \geq 10 \ \wedge \ x + y < \text{size}()$, where x and y are constrained variables. It is translated to the Cream constraint code shown in Figure 2.

5 Evaluation

We performed several experiments semi-automatically to evaluate the effectiveness and efficiency of our approach. One challenge for our experiments was that the constraint solving code should run in the same environment as that of the method under test because the constraints are written in terms of the names available to the method (e.g., formal parameters, fields of the receiver, and other methods of the class) and it should handle JML-extensions to Java (e.g., specification-only variables). Our solution was to manually inject constraint-solving code to the instrumented source code produced by the JML compiler.[2] We also extended both JET [6] and Cream [14]. JET is an automated unit testing tool that generates test cases randomly, and our extension was to implement the algorithm sketched in Section 4 as a new test data generation strategy,

[2]The JML compiler (`jmlc`) has an option (`--print`) to produce the instrumented source code before compiling it to bytecode.

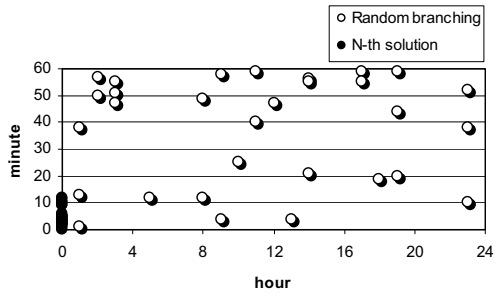


Figure 3. Diversity of solutions found for the constraint $0 \leq \text{hour} < 24 \wedge 0 \leq \text{minute} < 60$**.**

which essentially calls the injected constraint solving code as necessary.

The Cream extension was made to increase the diversity of generated test cases. Our initial experiments produced many duplicate or redundant test cases, and we shortly learned that this was caused by Cream's deterministic algorithm for enumerating solutions. If there are multiple solutions, Cream enumerates them in increasing order by always returning the smallest solution first. To remedy this problem, we introduced two techniques. The first technique, implemented without modifying the Cream framework, was to find and use the n-th solution. The second technique called a *random branching* introduced randomness in finding a solution by modifying the Cream framework. This modification explores the search space by bisecting the domains of variables. The original Cream explores by, at each step, selecting one variable, then exploring the rest of the search space by using only the first half of the domain of this variable; later, it looks for solutions in the other half. We changed this deterministic behavior by randomly choosing the half to explore first. This small modification greatly increased the diversity of the generated test cases, as shown in Figure 3, and improved the effectiveness of our approach.

We selected three classes for our experiments. As our approach offers benefits to methods with integral parameters, we selected classes of this characteristic.

1. `PINChecker`: This class stores, resets, and checks the validity of a personal identification number (PIN). The method parameters are a combination of objects and primitive data types.

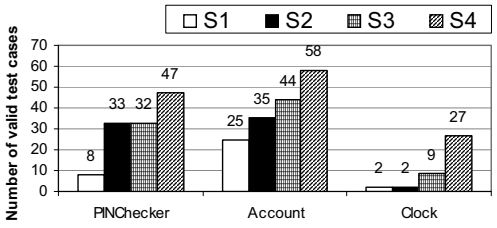
2. `Account`: This class represents a bank account and has methods such as `deposit`, `withdraw`, and `transfer` (see Figure 1). Most parameters are of integer type with non-negativeness constraints. Interestingly, using our approach we discovered an error in the `transfer` method—an overflow caused by adding a large number.

3. `Clock`: This class has a single method with a constraint like `0 <= hour && hour < 24`.

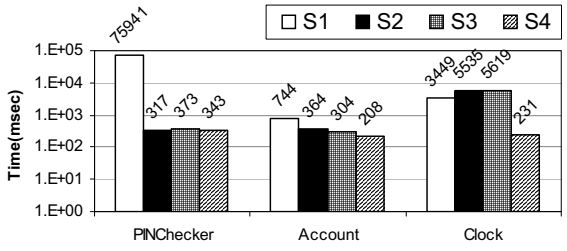
We evaluated the performance of four test generation strategies: the random strategy presented in [6] and three variations of our new approach (see Figure 4(a)). The variations correspond to the ways we used or modified the Cream framework. We first measured the number of non-duplicate, valid test cases generated by each strategy for each of the selected classes (see Figures 4(b)). As expected, constraint solving improved the effectiveness of random testing though the exact improvement varied widely depending on the characteristics of the classes. In particular, random branching is the most effective; for classes with non-trivial preconditions such as `PINChecker` and `Clock`, we noticed huge improvements in the numbers of generated test cases (i.e., 592% and 1331%, respectively). We next measured the time needed to generate a fixed number of non-duplicate, valid test cases (see Figure 4(c)). Again the improvements varied widely. The constraint strategy with random branching, for example, showed 22140%, 358%, and 1493% improvements for the three classes over the pure random strategy, giving an average of 7997% improvement. We ran the experiments on an AMD Turion™ 64X2 1.80 GHz with 2 GB of main memory.

Strategy	*Description*
S1	Pure random
S2	Constraint with first solution
S3	Constraint with n-th solution
S4	Constraint with random branching

(a) Test data generation strategies



(b) Average numbers of valid test cases generated for 100 attempts per method.



(c) Average time needed to generate 100 valid test cases per method.

Figure 4. Experimental results

6 Related Work

Previous work focused either on random testing (e.g., [6, 7, 8, 12]) or constraint solving (e.g., [1, 3]) in isolation, without taking an advantage of synergistic effects of both approaches. Some work also used meta-heuristic information to guide the search for valid test data (e.g., [9]); e.g., in genetic algorithms, call sequences that are likely to produce valid test data are selected and then made to evolve by applying genetic operations such as mutation and crossover [15].

The most closely related work is JML-TT [2], a specification animator for JML based on the constraint logic programming. It can execute methods leaving primitive parameters undefined or with ranges of values specified for them, showing a counter-example upon a specification violation. JML-TT can also generate test cases with boundary values. For this, it first extracts boundaries from preconditions and invariants. For example, a boundary test case for the `transfer` method of class `Account` could be `a1.transfer(1, a2)`, where `a1` and `a2` are objects of class `Account`, with balances zero and `Integer.MAX_VALUE`, respectively. Once a boundary test case is identified, it constructs needed objects (e.g., `a1` and `a2`) using the animator. However, this step is undecidable and thus may require a human assistance.

The `jmle` tool [10] is another specification animator for JML. It translates a JML specification to an executable Java implementation. The generated code relies on a constraint solver to simulate the specified behavior; i.e., the tool transforms a JML specification into a constraint satisfaction problem. For the approach to work, the specification should be detailed enough so that the constraint solver can reach the postcondition from the precondition. While `jmle` does not generate test cases, it would be possible to use some of its techniques to interpret a fixed set of method calls.

Jartege [12] is similar to JET in that it generates test data randomly and uses the runtime assertion checker as a test oracle procedure. There are also assertion-based random testing tools for other languages such as Eiffel [7].

7 Conclusion

We combined random testing with constraint solving to generate valid test data—test data that satisfies the precondition of the method under test. The key idea of our approach is first to generate random test data and then, if the generated test data does not satisfy the precondition, to solve constraints extracted from the precondition for parameters of finite domains such as integers. Our approach improves both the effectiveness of random testing from 6 to 13 times measured in the number of valid test cases generated and the efficiency from 4 to 221 times measured in the time needed to generate a given number of valid test cases.

Acknowledgment

Cheon's work was supported in part by NSF grants CNS-0509299 and CNS-0707874 and by the Department of Defense. Leavens's work was supported in part by NSF grant CNS-0808913.

References

[1] B. K. Aichernig and P. A. P. Salas. Test case generation by OCL mutation and constraint solving. In *Proc. of QSIC, Melbourne, Australia, September 19-20, 2005*, pages 64–71, 2005.

[2] F. Bouquet, F. Dadeau, B. Legeard, and M. Utting. Symbolic animation of JML specifications. In *ICFM*, volume 3582 of *LNCS*, pages 75–90. Springer-Verlag, July 2005.

[3] C. Boyapati, S. Khurshid, and D. Marinov. Korat: Automated testing based on Java predicates. In *ACM SIGSOFT ISSTA, Rome, Italy*, pages 123–133, July 2002.

[4] L. Burdy, Y. Cheon, D. Cok, M. Ernst, J. Kiniry, G. T. Leavens, K. R. M. Leino, and E. Poll. An overview of JML tools and applications. *International Journal on Software Tools for Technology Transfer*, 7(3):212–232, June 2005.

[5] Y. Cheon and G. T. Leavens. A simple and practical approach to unit testing: The JML and JUnit way. In *ECOOP*, volume 2374 of *LNCS*, pages 231–255. Springer-Verlag, June 2002.

[6] Y. Cheon and C. E. Rubio-Medrano. Random test data generation for Java classes annotated with JML specifications. In *SERP, Volume II, June 25–28, 2007, Las Vegas, Nevada*, pages 385–392, June 2007.

[7] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer. Experimental assessment of random testing for object-oriented software. In *ACM SIGSOFT ISSTA*, pages 84–94. ACM, 2007.

[8] C. Csallner and Y. Smaragdakis. JCrasher: an automatic robustness tester for Java. *Software—Practice and Experience*, 34(11):1025–1050, Sept. 2004.

[9] M. Harman and B. F. Jones. Search-based software engineering. *Info. & Software Technology*, 43(14):833–839, 2001.

[10] B. Krause and T. Wahls. jmle: A tool for executing JML specifications via constraint programming. In *FMICS*, volume 4346 of *LNCS*, pages 293–296. Springer-Verlag, 2006.

[11] G. T. Leavens, A. L. Baker, and C. Ruby. Preliminary design of JML: A behavioral interface specification language for Java. *ACM SIGSOFT Soft. Eng. Notes*, 31(3):1–38, Mar. 2006.

[12] C. Oriat. Jartege: A tool for random generation of unit tests for Java classes. In *ICQSA*, volume 3712 of *LNCS*, pages 242–256. Springer-Verlag, Sept. 2005.

[13] D. K. Peters and D. L. Parnas. Using test oracles generated from program documentation. *IEEE Transactions on Software Engineering*, 24(3):161–173, Mar. 1998.

[14] N. Tamura. Cream: Class library for constraint programming in Java. Available from `http://bach.istc.kobe-u.ac.jp/cream/`, as of January 2008.

[15] P. Tonella. Evolutionary testing of classes. In *Proc. of the ACM SIGSOFT ISSTA, Boston, MA*, pages 119–128, July 2004.

[16] E. Weyuker, T. Goradia, and A. Singh. Automatically generating test data from a boolean specification. *IEEE Transactions on Software Engineering*, 20(5):353–363, May 1994.

Properties of Machine Learning Applications for Use in Metamorphic Testing

Christian Murphy, Gail Kaiser, Lifeng Hu, Leon Wu
Department of Computer Science, Columbia University, New York NY 10027
{cmurphy, kaiser, lh2342, leon}@cs.columbia.edu

Abstract

It is challenging to test machine learning (ML) applications, which are intended to learn properties of data sets where the correct answers are not already known. In the absence of a test oracle, one approach to testing these applications is to use metamorphic testing, in which properties of the application are exploited to define transformation functions on the input, such that the new output will be unchanged or can easily be predicted based on the original output; if the output is not as expected, then a defect must exist in the application. Here, we seek to enumerate and classify the metamorphic properties of some machine learning algorithms, and demonstrate how these can be applied to reveal defects in the applications of interest. In addition to the results of our testing, we present a set of properties that can be used to define these metamorphic relationships so that metamorphic testing can be used as a general approach to testing machine learning applications.

1 Introduction

Making machine learning (ML) applications dependable presents a particular challenge because conventional software testing processes do not always apply: in particular, it is difficult to detect subtle errors, faults, defects or anomalies in the ML applications of interest because there is no reliable "test oracle" to indicate what the correct output should be for arbitrary input. The general class of software systems with no reliable test oracle available is sometimes known as "non-testable programs" [20].

One approach to testing such applications is to use a pseudo-oracle [6], in which multiple implementations of an algorithm process an input and the results are compared; if the results are not the same, then one or both of the implementations contains a defect. In the absence of multiple implementations, however, metamorphic testing [2] [8] [22] can be used to produce a similar effect: input can be modified in such a manner that it should produce the same output as the original, and if it does not, then a defect must exist. Of course, this can only show the existence of defects and cannot demonstrate their absence, since the correct output cannot be known in advance, but metamorphic testing provides a powerful technique to testing such "non-testable programs" by use of a built-in pseudo-oracle.

A challenge of metamorphic testing is to determine the so-called metamorphic relationships that can be used to transform an input such that its new output will be predictable, given the output produced by the original input. This generally requires domain knowledge and/or familiarity with the algorithm's implementation, and these relationships may not necessarily apply to other applications.

In this paper, we seek to create a taxonomy of metamorphic relationships that are applicable to input data of both supervised and unsupervised machine learning applications, including the inclusion and omission of data, permutation, and modification of numerical values. Our contribution is a set of properties that can be used to define these relationships so that metamorphic testing can be used as a general approach to testing machine learning applications.

Previously we have investigated approaches to testing such applications by considering properties of their data sets [14] and by using random testing [15]. In this paper, we first present our analysis of the metamorphic properties of MartiRank [9], a ranking implementation of the Martingale Boosting algorithm [12]. The result of this investigation is then used to guide the creation of metamorphic relationships that can be used in testing. We apply metamorphic testing to MartiRank, as well as to another machine learning algorithm, the anomaly-based intrusion detection system PAYL [19], and report our findings.

2 Background

2.1 Metamorphic testing

Metamorphic testing [2] [8] [22] is designed as a general technique for creating follow-up test cases based on existing ones, particularly those that have not revealed any failure, in order to try to find uncovered flaws. Instead of being an approach for test case selection, it is a methodology of reusing

input test data to create additional test cases whose outputs can be predicted. In metamorphic testing, if input x produces an output $f(x)$, a transformation function T can then be applied to the input to produce $T(x)$; this transformation is based on a metamorphic property of the function, such that the output $f(T(x))$ can then be predicted, based on the (already known) value of $f(x)$.

A classic example is the sine function. If we have built a function to compute sine, and for some selected input x we have computed $\sin(x) = y$, then we can create the test input $(x + 2\pi)$ and expect that $\sin(x + 2\pi)$ will also equal y, based on the metamorphic property of sine that $\sin(\alpha) = \sin(\alpha + 2\pi)$. Similarly, given $\sin(x) = y$, we can create the test input $-x$ and expect that $\sin(-x)$ should be $-y$, based on the metamorphic property of sine that $\sin(-\alpha) = -\sin(\alpha)$.

It is clear that this approach is very useful in the absence of an oracle. Regardless of the values of x and y, if $\sin(-x)$ does not equal $-\sin(x)$, then there must be a defect in the implementation of the sine function. Although the use of these simple identities for testing numerical functions is not unique to metamorphic testing [5], the approach can be used on a broader domain of any functions that display metamorphic relationships, including machine learning applications.

2.2 Related work

Applying metamorphic testing to situations in which there is no test oracle has been studied in great detail by Chen *et al.* [4]. Our work builds on theirs by applying metamorphic testing to a specific application domain (machine learning) and looking for the metamorphic relationships within those types of applications. Additionally, whereas their work has primarily focused on functions with simple numerical input domains [3], we are considering inputs that consist of larger (possibly alphanumeric) data sets, as a result of the types of applications we are investigating.

```
27,81,88,59,15,16,88,82,41,17,81,98,42, ..., 0
15,70,91,41, 5, 3,65,27,82,64,58,29,19, ..., 0
22,72,11,92,96,24,44,92,55,11,12,44,84, ..., 1
82, 3,51,47,73, 4, 1,99, 1,51,84, 1,41, ..., 0
57,77,33,86,89,77,61,76,96,98,99,21,62, ..., 1
...
```

Figure 1. Example of part of a data set used by supervised ML ranking algorithms

Although there has been much work that applies machine learning techniques to software engineering in general and software testing in particular (e.g., [1]), there has thus far been very little published work in the reverse sense: applying software testing techniques to ML applications that have no reliable test oracle. Orange [7] and Weka [21] are two of several frameworks that aid ML developers, but the testing functionality they provide is focused on comparing the quality of the results, and not evaluating the "correctness" of the implementations. Similarly, testing of intrusion detection systems [13] [17] has typically addressed quantitative measurements like overhead, false alarm rates, or ability to detect zero-day attacks, but does not seek to ensure that the implementation is free of defects.

2.3 Machine learning fundamentals

In general, data sets used in machine learning consist of a collection of *examples*, each of which has a number of *attribute* values and, in some cases, a *label*. The examples can be thought of as rows in a table, each of which represents one item from which to learn, and the attributes are the columns of the table. The label, if it exists, indicates how the example is categorized. In some cases a label of 1 is considered a *positive example*, and a 0 represents a *negative example*; without loss of generality, we only discuss these cases here. Figure 1 shows a small portion of a data set that could be used by such applications. The rows represent examples from which to learn, as comma-separated attribute values; the last number in each row is the label.

Supervised ML applications execute in two phases. The first phase (called the *training phase*) analyzes a set of *training data*; the result of this analysis is a *model* that attempts to make generalizations about how the attributes relate to the label. In the second phase (called the *testing phase*), the model is applied to another, previously-unseen data set (the *testing data*) where the labels are unknown. In a classification algorithm, the system attempts to predict the label of each individual example; in a ranking algorithm, the output of this phase is a ranking such that, when the labels become known, it is intended that the highest valued labels are at or near the top of the ranking.

Unsupervised ML applications also execute in training and testing phases, but in these cases, the training data sets necessarily do not have labels. Rather, an unsupervised ML application seeks to learn properties of the examples on its own, such as the numerical distribution of attribute values or how the attributes relate to each other. This model is then applied to testing data, to determine if the same properties exist. Data mining and collaborative filtering are two well-known examples of unsupervised learning.

2.4 Applications investigated

In this work we looked at two ML applications: MartiRank [9] and PAYL [19].

The development of MartiRank was commissioned by a company for potential future experimental use in predicting impending electrical device failures, using historic data of past device failures as well as static and dynamic information about the current devices. Classification in the binary

sense ("will fail" vs. "will not fail") is not sufficient because, after enough time, every device will eventually fail. Instead, a ranking of the propensity of failure with respect to all other devices is more appropriate.

In the training phase, MartiRank, which is a supervised ML algorithm, executes a number of "rounds". In each round the set of training data is broken into sub-lists; there are N sub-lists in the Nth round, each containing 1/Nth of the total number of positive labels. For each sub-list, MartiRank sorts that segment by each attribute, ascending and descending, and chooses the attribute that gives the best "quality". The quality of an attribute is assessed using a variant of the Area Under the Curve (AUC) [10] that is adapted to ranking rather than binary classification. The model, then, describes for each round how to split the data set and on which attribute and direction to sort each segment for that round. In the second phase, MartiRank applies the segmentation and sorting rules from the model to the testing data set to produce the ranking (the final sorted order).

```
1.0000,61,d
0.4000,32,a;1.0000,12,d
0.2500,18,d;0.5555,55,d;1.0000,41,d
```

Figure 2. Sample MartiRank model

Figure 2 shows a sample model. In the first "round", shown on the first line, all of the examples are sorted by attribute 61 (indicated by the "61") in descending order (indicated by the "d"). In the second round, shown on the second line, the result of the first round is then segmented. The first segment contains 40% of the examples in the data set (indicated by the "0.4000") and sorts them on attribute 32, ascending. The rest of the data set is sorted on attribute 12, descending. The two segments are then concatenated to reform the data set, which is then segmented and sorted according to the next line of the model, and so on.

We also investigated an intrusion detection system called PAYL. Many such systems are primarily signature-based detectors, and while these are effective at detecting known intrusion attempts and exploits, they fail to recognize new attacks and variants of old exploits. However, anomaly-based systems like PAYL are used to model normal or expected behavior in a system, and detect deviations of interest that may indicate a security breach or an attempted attack. PAYL was developed at Columbia University and is currently used in numerous real-world deployments.

As PAYL is an example of unsupervised machine learning (which is one reason why we chose to test it), its training data simply consists of a set of TCP/IP network payloads (streams of bytes), without any associated lables or classificiation. During its training phase, it computes the mean and variance of the byte value distribution for each payload length in order to produce a model; Figure 3 shows an example of such a distribution [19]. During the second

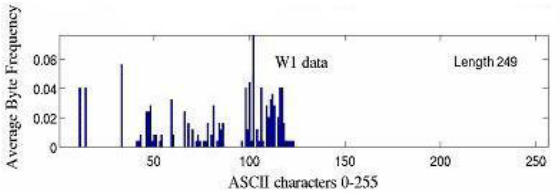


Figure 3. Sample payload byte distribution

("detection") phase, each incoming payload is scanned and its byte value distribution is computed. This new payload distribution is then compared against the model (for that length) using the Mahalanobis distance, which is a way of comparing two sets of data but unlike Euclidean distance does not depend on the scale of the values; if the distribution of the new payload is above some threshold of difference from the norm, PAYL flags the packet as anomalous and generates an alert. PAYL may also raise an alert in other circumstances, for instance if the payload length had never been seen before in the training data.

3 Approach

We previously reported on our testing of ML ranking applications (MartiRank and SVM-Light [11]) in which we developed test cases by analyzing the problem domain, analyzing the algorithms as defined in pseudo-code, and analyzing the runtime options [14]. This then allowed us to devise equivalence partitions that served as guidelines for the creation of datasets using random testing [15].

We then went back to these applications and used our knowledge of the algorithms to identify metamorphic relationships (previously unpublished) that would give us another way of testing such applications in the absence of an oracle. These properties are described in Section 4.

Once we had enumerated and categorized the different types of metamorphic properties, we used these principles in our testing. We first tested an implementation of MartiRank, and then sought to also apply these to the anomaly-based intrusion detection system PAYL, on which we conducted metamorphic testing using the same guidelines. Section 5 discusses the results of our testing.

4 Metamorphic properties

We begin by describing our observations of the metamorphic properties of MartiRank [9]. We first considered metamorphic relationships that should not affect the output: either the model that is created as a result of the training phase, or the ranking that is produced at the end of the testing phase. For the training phase, if training data set input D produces model M, then we looked for transformation

functions *T* so that input *T(D)* would also produce model *M*. Additionally, if testing data set input *K* and model *L* produce ranking *r(K, L) = R*, then we looked for transformation functions *T* so that the combinations *r(T(K), L)*, *r(K, T(L))* and *r(T(K), T(L))* all produce *R* as well.

Based on our analysis of the MartiRank algorithm, we noticed that it is not the actual values of the attributes that are important, but it is the *relative* values that determine the model. Adding a constant value to every attribute, or multiplying each attribute by a positive constant value, should not affect the model because the model only concerns how the examples relate to each other, and not the particular values of the examples' attributes. The model declares which attributes to sort to get the best ordering of the labels; in Figure 1, if the values in any column were all increased by a constant, or multiplied by a positive constant, then the sorted order of the examples would still be the same, thus the model would not change. Additionally, applying a given model to two data sets, one of which has been created based on the other but with each attribute value increased by a constant, would generate the same ranking, based on the same line of reasoning. Thus, MartiRank exhibits metamorphic properties that we can classify as both **additive** and **multiplicative**: modifying the input data by addition or multiplication should not affect the output.

It should also be the case that changing the order of the examples should not affect the model (in the first phase) or the ranking (in the second). As MartiRank is based on sorting, in the cases where all the values for a given attribute are distinct, it is clear that the sorted order will still be the same regardless of the original input order. Thus, MartiRank also has a **permutative** metamorphic property, albeit only limited to certain inputs.

We then considered metamorphic relationships that would affect the output, but in a predictable way. For the training phase, if training data set input *D* produces model *M*, then we looked for transformation functions *T* so that input *T(D)* would produce model *M'*, where *M'* could be predicted based on *M*. Additionally, if testing data set input *K* and model *L* produce ranking *r(K, L) = R*, then we looked for transformation functions *T* so that *r(T(K), L)*, *r(K, T(L))* and *r(T(K), T(L))* all can be predicted based on *R*. Keep in mind that in order to perform testing, we need to be able to have a predictable output based on *R* because we cannot know it in advance otherwise, since there is no test oracle.

We mentioned above that multiplying all attributes by a positive constant should not affect the model. On the other hand, mulitplying by a negative constant clearly would have an effect, because sorting would now result in the *opposite* ordering. The effect on the MartiRank model, however, could easily be predicted, because the model not only specifies which attribute to sort on, but which direction (ascending or descending) as well. Consider that, if one were to sort a group of numbers in ascending order, then multiply them all by a negative constant, and sort in descending order, the original sorted order would be kept intact. In MartiRank, if in the original data set a particular attribute is deemed to be the best one to sort on, and a new data set is created by multiplying every attribute value by a negative constant, then that particular attribute will still be the best one to sort on, but in the opposite direction. The only change to the model will be the sorting direction. Thus, MartiRank displays an **invertive** metamorphic property, wherein it is possible to predict the output based on taking the "opposite" of the input. Like the permutative property, this property only holds in the case where all values are distinct, however.

This invertive property can also be seen in the testing phase. For data set input *K*, we define *K'* as its inverse, *i.e.* all attribute values multiplied by a negative constant. For model *L*, we define *L'* as its inverse, *i.e.* the sorting directions all changed. We also define *R = r(K, L)* as the ranking produced on data set *K* and model *L*, and *R'* as the inverse ranking, where the examples are ranked in "backwards" order. Based on the explanation above, we can expect that if *r(K, L) = R*, then *r(K', L')* is also equal to *R*, because sorting the positive values ascending will yield the same ordering as sorting the negative values descending. It follows, then, that *r(K', L)* and *r(K, L')* should both be equal to *R'*, in which the ranking is the same but in the opposite direction.

Furthermore, once we know the model, it is easy to add an example to the set of testing data so that we can predict its final place in the ranking. Take, for example, the model shown in Figure 2. In the first round, it sorts on attribute 61 in descending order; if we add an example to a testing data set such that the example has the greatest value in attribute 61, it will end up at the top of the sorted list. In the second round, the model sorts the top 40% (which would include our added example) on attribute 32 in ascending order; if we modify our added example so that it has the smallest value for attribute 32, it will stay at the top of the list. And so on. Knowing the model, we can thus construct an example, add it to the data set, and expect it to appear first in the ranking. We can thus say that MartiRank has an **inclusive** metamorphic property, meaning that a new element can be included in the input and the effect on the output is predictable. Similarly, MartiRank also shows an **exclusive** metamorphic property: if an example is excluded from the testing data, the resulting ranking should stay the same, but without that particular example, of course.

5 Case studies

As a result of our investigation of MartiRank, we have identified six metamorphic properties of supervised ML applications: additive, multiplicative, permutative, invertive, inclusive, and exclusive. We have also discovered that these

properties exist in another ranking algorithm, SVM [18], as well; due to space constraints, those findings are not discussed here but can be found in our tech report [16]. Although we have only considered ranking algorithms thus far, we believe that classification algorithms would display the same properties because of the similarity of the algorithms in terms of the ways in which they treat the data; this is left as future work. Following that analysis, we conducted metamorphic testing using those properties.

5.1 Metamorphic testing of MartiRank

After identifying the metamorphic properties of MartiRank, we constructed corresponding test cases and were able to detect a defect in the implementation. Another of its invertive properties is that if all of the labels in the training data are negated (multiplied by -1), the final ranking of the testing data should be the same but in opposite order from the original, since what was the "worst" would now be considered "best". However, as the particular implementation we were testing was designed specifically to rank the likelihood of device failures, the developers never considered the case in which the labels in the training data (which represented the number of failures over a given period of time) would be negative. During metamorphic testing, the implementation produced inconsistent results when a negative label existed, and we confirmed this bug first with a simple toy data set and then upon inspection of the code, in which a logical flaw existed in the way the examples were being segmented during training.

5.2 Analysis of PAYL

We next sought to determine whether the properties we used to guide metamorphic testing of MartiRank could also be applied to another ML application. The application we chose was PAYL [19], an anomaly-based intrusion detection system.

Because the model generated by PAYL in the training phase represents the distribution of byte values in the TCP/IP payload (see Figure 3), it is clear that it exhibits the additive and multiplicative properties. Adding a constant value to each byte would shift the distribution, and multiplying by a constant would stretch it. Therefore, it would be easy to predict the effect on the model. Additionally, the categorization (as anomalous or not) of a packet in the testing phase would not change if it, too, had its bytes modified in the same manner.

Much of our analysis of PAYL focused on its permutative properties, primarily because some attackers may try to hide a worm or virus by permuting the order of the bytes, so as to trick a signature-based intrusion detection system. Of course, the model created by PAYL does not consider the order of the bytes, only their distribution, so a permutation should still result in the same model.

PAYL also has an invertive property. An "inverse" of the distribution can be obtained by subtracting each byte value from the maximum (255, or 0xFF), so that frequently-seen values become less frequent, and vice-versa. If the same treatment is applied to the payloads in the testing data, then the same alerts should be raised, since these values will still appear to be anomalous.

Aside from considering the distribution of byte values in creating its model, PAYL also considers the existence (or absence) of payloads of certain lengths, and thus certainly has inclusive metamorphic properties. For instance, consider a model that generates an alert on a new payload because its length had never before been seen. If the particular payload were then included in the training data, it should no longer be considered anomalous. We would similarly expect PAYL to have exclusive metamorphic properties: if all payloads of a certain length were removed from the set of training data, then any messages of that length in the testing data would thus be considered anomalous because they had not previously been seen.

5.3 Metamorphic testing of PAYL

We then conducted testing of PAYL by using data sets generated via these metamorphic relationships. By using the exclusive metamorphic property, we were able to detect two defects in PAYL. We started with training data that had payloads of various sizes, including 274 bytes, and created a model that was applied to a set of testing data, which also included a payload of 274 bytes; PAYL raised no alerts. We then removed all payloads of 274 bytes from the training data and applied the model to the same (unmodified) testing data, expecting that the payload of 274 bytes in the testing data would cause PAYL to raise a "length-never-seen-before" alert. However, PAYL raised an anomaly alert for the payload of length 274, even though there was no payload of that length in the training data. An alert was correctly being raised, but it was the wrong type.

Additionally, PAYL unexpectedly raised *both* anomaly alerts *and* "length-never-seen-before" alerts for payloads of 1448 bytes, which theoretically should never happen (since it can only be anomalous if that length had actually been seen before). Upon further investigation, we determined that PAYL actually should have raised the "length-never-seen-before" alert from the first set of training data, since there were no payloads of that length. So not only were the alerts not being raised in the first place, but false positives were then being raised in the second.

Our key result, though, was that we were able to verify that PAYL exhibits the same six metamorphic properties as MartiRank, and then use these properties to drive metamor-

phic testing and find important defects in PAYL.

6 Conclusion and future work

We have identified six metamorphic properties that we believe exist in many machine learning applications: additive, multiplicative, permutative, invertive, inclusive, and exclusive. Although these are likely not the *only* metamorphic properties that can exist in a machine learning algorithm, they provide a foundation for determining the relationships and transformations that can be used for conducting metamorphic testing, which we have shown to reveal defects in the applications of interest.

Further investigation would involve applying these metamorphic properties to other, larger ML applications, and looking to classify other properties. Additionally, as we have defined our properties independent of the actual numerical values used in the data sets, future work could consider how to initially create new data sets such that further application-specific metamorphic properties can also be revealed.

We have found metamorphic testing to be an efficient and effective approach to testing ML applications. We hope that our findings here and the identification of metamorphic properties help others who are also concerned with the quality of non-testable programs.

7 Acknowledgments

The authors would like to thank T.Y. Chen, Marta Arias, Hila Becker, Gabriela Cretu, Phil Gross, and David Waltz for their assistance. Murphy, Kaiser, and Wu are members of the Programming Systems Lab, funded in part by NSF CNS-0717544, CNS-0627473, CNS-0426623 and EIA-0202063, and NIH 1 U54 CA121852-01A1.

References

[1] T. J. Cheatham, J. P. Yoo, and N. J. Wahl. Software testing: a machine learning experiment. In *Proc. of the ACM 23rd Annual Conference on Computer Science*, pages 135–141, 1995.

[2] T. Y. Chen, S. C. Cheung, and S. Yiu. Metamorphic testing: a new approach for generating next test cases. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, 1998.

[3] T. Y. Chen, F.-C. Kuo, T. H. Tse, and Z. Q. Zhou. Metamorphic testing and beyond. In *Proc. of the International Workshop on Software Technology and Engineering Practice (STEP)*, pages 94–100, 2004.

[4] T. Y. Chen, T. H. Tse, and Z. Q. Zhou. Fault-based testing without the need of oracles. *Information and Software Technology*, 44(15):923–931, 2002.

[5] W. J. Cody Jr. and W. Waite. *Software Manual for the Elementary Functions*. Prentice Hall, 1980.

[6] M. D. Davis and E. J. Weyuker. Pseudo-oracles for non-testable programs. In *Proc. of the ACM '81 Conference*, pages 254–257, 1981.

[7] J. Demsar, B. Zupan, and G. Leban. Orange: From experimental machine learning to interactive data mining. [www.ailab.si/orange], Faculty of Computer and Information Science, University of Ljubljana.

[8] A. Gotleib and B. Botella. Automated metamorphic testing. In *Proc. of 27th annual international computer software and applications conference (COMPSAC)*, pages 34–40, 2003.

[9] P. Gross et al. Predicting electricity distribution feeder failures using machine learning susceptibility analysis. In *Proc. of the 18th Conference on Innovative Applications in Artificial Intelligence*, 2006.

[10] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36, 1982.

[11] T. Joachims. *Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1999.

[12] P. Long and R. Servedio. Martingale boosting. In *Proc. of the 18th Annual Conference on Computational Learning Theory (COLT)*, pages 79–84, 2005.

[13] P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman. An overview of issues in testing intrusion detection systems. Tech. Report NIST IR 7007, National Institute of Standard and Technology.

[14] C. Murphy, G. Kaiser, and M. Arias. An approach to software testing of machine learning applications. In *Proc. of the 19th international conference on software engineering and knowledge engineering (SEKE)*, pages 167–172, 2007.

[15] C. Murphy, G. Kaiser, and M. Arias. Parameterizing random test data according to equivalence classes. In *Proc of the 2nd international workshop on random testing*, pages 38–41, 2007.

[16] C. Murphy, G. Kaiser, and L. Hu. Properties of machine learning applications for use in metamorphic testing. Technical Report cucs-011-08, Dept of Computer Science, Columbia Univ, 2008.

[17] J. P. Nicholas, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson. A methodology for testing intrusion detection systems. *IEEE Transactions on Software Engineering*, 22(10):719–729, 1996.

[18] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[19] K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection. In *Proc. of Recent Advances in Intrusion Detection (RAID)*, Sept. 2004.

[20] E. J. Weyuker. On testing non-testable programs. *Computer Journal*, 25(4):465–470, November 1982.

[21] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd Edition*. Morgan Kaufmann, 2005.

[22] Z. Q. Zhou, D. H. Huang, T. H. Tse, Z. Yang, H. Huang, and T. Y. Chen. Metamorphic testing and its applications. In *Proc. of the 8th International Symposium on Future Software Technology (ISFST 2004)*, 2004.

Fault Injection Testing of User-space File Systems Using Traditional and Aspect-based Techniques

Jonathan Hittle, Sudipto Ghosh
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523, USA
{*hittlej, ghosh*}*@cs.colostate.edu*

Abstract

File systems are part of larger, complex systems and are used in large-scale configurations that are built on a large number of disks. File systems rely on other sub-systems, such as I/O and memory management that can fail, but they are often implemented as if the other sub-systems will not fail. Testing file systems for proper operation in large-scale configurations is a prime concern for vendors of storage servers. In this paper, we examine the feasibility of using fault injection testing on user-space file systems, both with traditional code insertion techniques and with aspects.

Keywords: *Aspect-oriented programming, fault injection testing, file systems testing, user-space file systems*

1. Introduction

File systems are susceptible to failures in the underlying subsystems, such as disks, I/O bus, and memory management. Fault injection testing can be used to make file systems resilient to sub-system failures. The overall goal of our research is to evaluate and compare the costs and benefits of performing fault injection testing on user-space and kernel file systems using both traditional code insertion and aspect-oriented programming techniques. In this paper, we report results from fault injection testing using traditional code insertion and aspects on the `ext2` FUSE file system. We compare the two approaches in terms of the complexity involved in implementing the fault injection code.

Current approaches (e.g., [3, 4]) to fault injection on file systems use procedural or object-oriented techniques to insert code in the kernel space where engineering costs can be high. They insert a specialized layer of code between the module under test and the underlying subsystems (e.g., between the file system and memory management). This keeps the fault injection modular but it does not allow for fault injection within the file system module. As an example, injecting faults into a file system's inode caching code would require changes within the file system, and cannot be accomplished without adding code to the file system. Using aspect-oriented programming [2] instead of traditional code insertion can provide the means to performing testing at all levels within a file system (intra-function, intra-module, and inter-module).

Performing fault injection on user-space file systems instead of kernel-space file systems can potentially lower the time and engineering costs. With the advent of user-space file systems through frameworks such as FUSE[1], we now have options for fault injection testing without the overhead of development and testing within the kernel. We can inject faults between the file system and the storage media through read faults, write faults, and whole-disk faults. We used the FUSE implementation of the Linux `ext2` file system and selected the set of faults from Nagaraja et al. [3]. We wrote the corresponding fault routines and inserted them using both procedural and aspect-oriented techniques.

The remainder of this paper is organized as follows. Section 2 provides background and related work. Section 3 describes the proposed user-space fault injection approach using both procedural and aspect-oriented methods. Section 4 discusses the feasibility of the approach. Section 5 presents the conclusions and outlines directions for future work.

2. Background

File systems: File systems provide the structure for organizing, storing, and retrieving data in a computer system. Windows, Linux, and most Unix-like operating systems provide an infrastructure for allowing multiple types of file systems. The generic term for the infrastructure is a Virtual File System (VFS) [5]. It provides a generic interface to the kernel and system calls, such as `open`, `read`,

[1] See `http://fuse.sourceforge.net/`.

and `write`. It also routes the requests to the correct kernel module that understands the file system. That module then interacts with the cache and I/O managers to read and write data stored in a formatted manner on a disk.

Traditionally, file systems has been kept in kernel-space, where interactions with the cache manager and the I/O system are simple and the execution overhead small. To work within this model, a user-space file system has a module that lives in kernel-space that can receive requests from the VFS, relay them to the user-space process handling the I/O necessary for the mounted file system, and return the responses back to the VFS. With the FUSE module acting as the intermediary, the kernel is oblivious to the fact that the file system is actually being handled by a user-space process.

This concept of user-space file systems is not widely seen in practice. User-space file systems have the appeal of being in an environment with which most programmers are familiar, including debugging tools which are typically more feature-rich than their kernel counterparts.

Fault injection testing: The goal of fault injection testing is to see how a software component reacts to a failure condition [1]. Fault injection testing on file systems has focused on disk, I/O bus, and memory failures. Nagaraja et al. [3] used the SCSI faults in Table 1 in their Mendosa infrastructure. Column one shows the types of faults that are possible. This table covers only I/O related faults. Other faults are possible too, since the file system depends on things other than I/O, such as the cache manager and the virtual memory subsystem, and faults can be injected into any of those interactions. Faults can also be injected within the file system, such as inode cache faults and locking faults, but this is outside the scope of this paper.

Table 1. SCSI Faults from Nagaraja et al. [3].

Fault	Characteristic	OS Masking of Fault	Our fault
Disk Hang	Sticky	Unmasked	Disk loss
Disk offline	Sticky	Unmasked	Disk loss
Power Failure	Sticky	Unmasked	Disk loss
Read Fault	Sticky	Unmasked in Linux	Read failure
Write fault	Sticky	Unmasked in Linux	Write failure
Timeout	Transient	Unmasked	Timeout
Parity errors	Transient	Masked	N/A
Bus busy	Transient	Masked	N/A
Queue full	Transient	Masked	N/A

The "Characteristic" column conveys the idea of the *stickiness* of the fault, that is, whether the fault is permanent or not. In the case of disk or block failures, if the device has either failed in such a way that it is permanently failed or it is unreliable, then the failure is treated as permanent.

Column three lists whether the fault is *masked* or not. As Nagaraja et al. [3] point out, faults can either be masked from the file system or unmasked. A masked fault is one which will not be seen by the module under test (the file system driver). For example, the I/O subsystem will retry when a SCSI queue is full or when the bus is busy, and ECC memory will mask parity errors. Table 1 shows that most faults are seen by the file system. Issues related to the SCSI controller and bus are handled by the lower software layers. The faults used in this research are a subset of those represented in the table. The implemented faults shown in column four do not contain the faults that are masked from the file system. Moreover, the three disk failure faults are rolled into one fault. The faults are implemented to have the sticky characteristic.

In their IRON file system work, Prabhakaran et al. [4] used fault injection to test file systems for their tolerance to I/O errors. A key idea is that a file system should not fail an entire disk because of one simple failure. For example, it should instead handle single-block failures as single-block failures. Single block failure represents a recoverable situation, requiring only that the bad block be remapped.

3. Approach

We used Aspect-Oriented C[2] to implement the four faults shown in Table 1. We wrote the actual fault injection routines in a common file that could be reused for both the approaches. We modified the `libext2fs`-based[3] FUSE implementation of the `ext2` file system by using traditional code insertion techniques to call the fault injection routines within the `read` and `write` paths. Next, we wrote aspects to do the same code insertion within the FUSE `ext2` file system without changing any of the FUSE `ext2` code, and wove those aspects into the code. Finally, we used the `iozone`[4] benchmark to generate a load on the file system to test that the fault injection technique was working as expected. We demonstrated that all the faults were injected into the FUSE `ext2` file system using both approaches.

At the core of the fault injection testing technique are the fault injection routines: `doBlkFIT()` and `doDiskFIT()` are the entry points for the block-based and disk-based fault injection respectively. Both routines dynamically inject faults based on the number of operations that have occurred. Pseudo-code is given in Figure 1.

Since fault injection is done based on I/O numbers, we first increment the count of I/O operations. We check the sticky faults of a failed block or a failed disk. We also check the I/O number to see if a new failure needs to occur. The faults are checked in ascending order by the frequency with which they are planned to be injected. Disk failures happen once every one thousand I/O operations. Timeout failures happen once every twenty I/O operations. Disk block failures happen once every ten I/O operations. The `libext2fs` code contains the core code for handling I/O. For the traditional fault injection, the *faulty* code was

[2] See `http://research.msrg.utoronto.ca/ACC/`.
[3] See `http://e2fsprogs.sourceforge.net/`.
[4] See `http://www.iozone.org/`.

```
    doBlkFIT(blockNo) {
        ioCount := ioCount + 1
        if (blockNo has been failed)
            return I/O error
        if (disk has been failed)
            return no device failure
        if (ioCount = disk failure I/O)
            change disk state to failed
            return I/O error
        if (ioCount = timeout failure)
            return timeout error
        if (ioCount = block failure)
            change block state to failed
            return I/O error
        return success
    }
    doDiskFIT() {
        ioCount := ioCount + 1
        if (disk has been failed)
            return no device failure
        if (ioCount = disk failure)
            change disk state to failed
            return I/O error
        return success
    }
```

Figure 1. Fault Injection Routines.

inserted into the core code. For the aspect-oriented fault injection, aspects were written that were woven into the entry points in the core code.

```
error := read(
    ioController,
    physical block number,
    input buffer);
if (error)
    return error
```

Figure 2. Original Read Path.

Figure 2 shows the original logic of the `libext2fs` routine `load_buffer()`, which is at the core of `libext2fs`'s `read` path. The file system's I/O controller issues the actual `read` and places the data into the given buffer. An error causes an immediate exit from `load_buffer()`; on success `load_buffer()` does more processing and the data is passed back to the caller of the `read()` system call.

Figure 3 illustrates how we modified the `libext2fs` routine `load_buffer()` to do the traditional code insertion fault injection. We add a call to the fault injection routine. This alters the code path so that if a fault injection is to occur, the routine that does the read into the buffer is not actually called. This is

```
error := doblkFIT(physical block number);
if (error)
    return error;
error := read( ioController,
    physical block number,
    input buffer);
if (error)
    return error;
```

Figure 3. Modified Read Path.

slightly different from traditional code insertion, which would place a layer of code between `load_buffer()` and `io_channel_read_blk()`, which would encapsulate the original direct call to `io_channel_read_blk()` without changing any arguments. However, that is an engineering concern, and what we did here is sufficient for the purposes of this study. We point out this difference to note that this insertion could be done in a modular fashion which would not require any actual modification to `load_buffer()`. We understand this difference and do not consider the chosen implementation to be representative of modular traditional code insertion.

In the modified code flow, the `doBlkFIT()` routine is called before performing the I/O. If a fault is to be injected, then the corresponding error code is returned, and `load_buffer()` passes the error back to its caller. If no fault is to be injected, then `doBlkFIT()` returns success, and `load_buffer()` continues with the original path, passing control to `io_channel_read_blk()`. The write block, timeout, and disk faults are all injected using similar means in the routines `ext2fs_file_write()`, `ext2fs_file_read()`, and `ext2fs_bmap()`. Code for `ext2_file_flush()` is not shown here for lack of space.

```
errcode_t around():
execution(errcode_t ext2fs_file_write(...))
|| execution(errcode_t ext2fs_file_read(...))
|| execution(errcode_t ext2fs_bmap(...)) {
      errcode_t error;
      error := do_dsk_FIT();
      if (error)
          return error;
      return proceed();
}
```

Figure 4. Disk Fault Injection Aspect.

Figure 4 shows the disk fault injection aspect, which performs the aspect-oriented code insertion fault injection testing. It is structured similar to the code used in the traditional code insertion case. The pointcut specifies that the advice is to be placed around the execution of

`ext2fs_file_write()`, `ext2fs_file_read()`, and `ext2fs_bmap()`. The fault injection routine is called, and any injected error is returned. If no error is returned, then the original call is completed and the result returned.

Iozone issues reads, writes, and seeks within a file. The I/O that these activities cause was used to validate that faults were being injected in both the approaches. Since `ext2` is a file system that is not intended to handle disk faults, all the failures caused iozone to stop execution. As `ext2` failed on each fault one by one, we removed the faults from our set until the set was empty. This way we verified that all the injected faults caused FUSE `ext2` to fail.

4. Results

We evaluated the traditional code insertion technique first. In the case of the timeout and write faults, the file system ended up in a corrupted state. Essentially, the file was corrupted such that attempting to delete it would cause `ext2` to attempt to clear a block with an invalid number, causing the process handling `ext2` to crash. Clearly, this is an example of fault injection testing revealing a failure, since data corruption is never an acceptable behavior. Both the read and whole disk faults caused iozone to stop. However, neither left the disk corrupted. All that was needed to try iozone again was to unmount and remount the file system. FUSE uses a single process for a file system mount and the fault injection routines and associated state are stored in that per-mount process. Therefore, unmounting and remounting the file system has the side effect of restarting the fault injection. This is an example of behavior which must be considered within the context of the file systems' intended application to know whether it is acceptable or not. Overall, this indicates that code insertion fault injection testing can be applied to user-space file systems.

Next, we evaluated the aspect-oriented code insertion technique. The faults were injected in the same order, with the same results. This shows that aspects can be used for fault injection testing of user-space file systems as effectively as traditional code-insertion techniques.

The entire user-space effort took less than 10 hours. Half of that time was spent studying the `ext2` and the `libext2fs` code. One hour was spent developing the fault injection routines. For traditional code insertion, it took 50 minutes to go from instrumented code injecting all four faults to having encountered and removed all of them. Also, the code insertion was done as efficiently using aspects if the overhead of learning how to use the ACC language and ACC tools is not included. The time needed for familiarity with ACC was two hours. The time it took to go from code with all aspects woven into the system to having hit all of the faults was 40 minutes. It is expected that part of the decrease in time is due to familiarity with the process. Therefore, no difference in efficiency was seen.

It is difficult to compare the two approaches from this study because (1) they rely on common code, (2) the number of faults used is small, (3) only one file system implementation was used for this study, and (4) they were both developed by the same person. The first issue is of low priority, since it is fair to compare the two approaches using common code where appropriate. The second issue needs to be addressed by expanding the library of faults. This will lead to finding those faults which can be injected into user-space file systems that cannot be injected into kernel-space file systems and vice versa. The third issue can be addressed by using more file systems. This will be difficult to do in general because there are few file systems that are available in both user-space and kernel-space versions. This makes it difficult to generalize the results of the study. The fourth issue needs to be addressed by having multiple developers separately using the two approaches.

5. Conclusions and Future Work

We demonstrated that fault injection testing can be used for user-space file systems using both code-insertion and aspect based fault techniques. We inserted faults for read block, write block, whole disk loss, and disk timeout faults. To explore the generalized application of these results, more file systems and faults must be included.

We will investigate how well the approach transfers into kernel-space, and also check if there are faults that can be injected in kernel-space but not in user-space. We will enhance the fault library to include faults such as memory load/store faults, memory space faults, and buffered I/O faults. We will carry out studies to see if developers can use fault-injection testing of a file system in user-space to more efficiently improve the quality of a file system.

References

[1] M. C. Hsueh, T. K. Tsai, and R. K. Iyer. Fault Injection Techniques and Tools. *IEEE Computer*, 30(4):75–82, April 1997.

[2] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingier, and J. Irwin. Aspect Oriented Programming. In *Proc. of the European Conf. on Object-Oriented Programming, LNCS 1241*, Finland, June 1997.

[3] K. Nagaraja, X. Li, R. Bianchini, R. P. Martin, and T. D. Nguyen. Using fault injection and modeling to evaluate the performability of cluster-based services. In *USENIX Symp. on Internet Technologies and Systems*, 2003.

[4] V. Prabhakaran, L. N. Bairavasundaram, N. Agrawal, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Iron file systems. In *Proc. of the 20th ACM Symp. on Operating Systems Principles*, pages 206–220, UK, October 2005.

[5] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts 7th Edition*. John Wiley and Sons, USA, 2005.

Evaluation of Personalized Information Systems: Application in Intelligent Transport System

M.Soui[1], C.Kolski[1], M.Abed[1], G. Uster[2]

[1] LAMIH-UMR CNRS 8530, University of Valenciennes and Hainaut-Cambrésis, Le Mont Houy, 59313 Valenciennes cedex 9, France
{Makram.Soui, Christophe.Kolski, Mourad.Abed}@univ-valenciennes.fr

[2] The French national institute of research for transport and security, INRETS, Villeneuve d'Ascq, France
guillaume.uster@inrets.fr

***Abstract*- Thanks to the emergence of the Personalized Information System (PIS), it becomes possible to supply the user only with the pertinent information that directly interests him/her and suits his/her preferences. However, we need beforehand to evaluate these systems in real situations. In this paper, we point out the insufficiencies in the evaluation of the PIS. Then, we propose a new evaluation method for PIS. Finally, we describe the evaluation results of a demonstrator developed during a project called MouverPerso. This demonstrator was tested among a group of subjects in the University of Valenciennes.**

Index Terms- **Evaluation, Human-Computer Interaction (HCI), Intelligent Transport System (ITS), Personalized Information System (PIS)**

I. Introduction

Nowadays, the information systems, addressed to the users and mainly in the field of transport, tend to be more and more personalized. In the midst of a universal project labelled Intelligent Transport System (ITS), researchers' primary preoccupation is to provide personalized information for the public transport users. In fact, the PIS is a system which has the capacity to be adapted to the user taking in consideration his/her preferences [1]–[2]–[3]. For example, in transport field, the traveler hopes to have at his/her disposal only some information, just what he/she is directly interested in [4]–[5]. In addition, the traveler may have access to a reliable, multi-modal and personalized information using various supports (PC, PDA, mobile phone, etc.) [6]. Though, there are different methods and approaches to conceive PIS systems, to our knowledge and at the present point of research, we notice a lack of methods to evaluate the personalization quality of PIS. This article is made of three principal parts: at first we focus on the insufficiencies concerning the evaluation of PIS. The second part is devoted to describe the basic principles of a proposed method that permits the evaluation of PIS. The third and last part is meant to describe the evaluation results of the demonstrator developed during a project called MouverPerso.

II. Insufficiency in The pis Evaluation

The evaluation of the interactive system has been a recurrent problem since the last three decades. To ameliorate the quality of human machine interaction many studies were oriented towards the evaluation of the interactive system from different angles and view points. Concerning this subject we should mention that several papers and works had defined the basic principals of the evaluation as well as the methods used in this evaluation [7]–[8]–[9]–[10]–[11]–[12]–[13]–[14]. We notice that the focal point in these works was on the utility and usability while the dimension of personalization was neglected. Nowadays, the PIS users are facing many difficulties to interact with the badly studied or evaluated personalized systems that do not always answer their needs. Due to the complexity of personalized system interface, new criteria and methods are needed to evaluate the human machine interaction. In this context, we propose a method which is based on the evaluation criterion of PIS detailed in [15] and on the explicit intervention of the user who fills up questionnaires.

III. Proposition of a Method for Evaluating PIS

The process of this evaluation is illustrated in the fig. 1. To give a clear structure to this proposition we use the SADT formalism; a well known in software engineering and in human machine interaction. This method is made up of three phases. According to the SADT formalism, we find in the box A0 (the box in top of diagram) the general objective that consists in the evaluation of PIS. To reach this target we have decomposed it into three sub-targets presented in the boxes A1, A2 and A3 (preparation, evaluation and analysis).

- Phase A1: it represents the preparation stage wherein the evaluator chooses the representative tasks on which the evaluation will be based. The evaluator also prepares two types of documents necessary to the evaluation. The first is a general questionnaire including general information about the users and the second is an index-card that includes the definition of every criterion and the parts which the user fills up during the experimentation. We distinguish seven global criteria, detailed in [15].

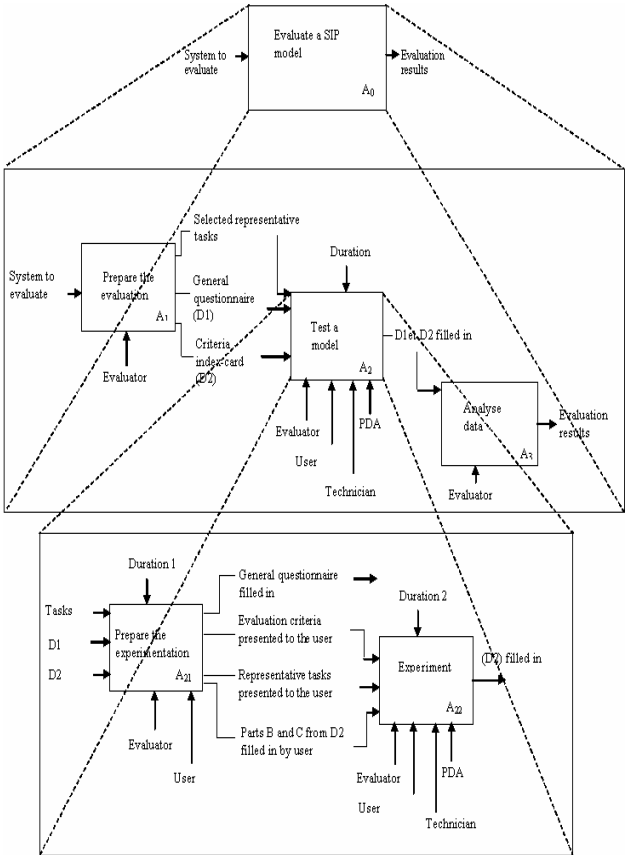


Fig. 1. Evaluation phases of the proposed method.

- Phase A2: it represents the stage of testing the model; it is made of two sub-phases (A21 and A22):
- Sub-phase A21: it represents the pre-experimental stage; the evaluator presents to the users the tasks already chosen in addition to the evaluation criteria and their definition. Then the evaluator asks them to select the most important criteria according to their needs and/or preferences. The users must attribute to every selected criterion a weight representing the importance accorded to this criterion.
- Sub-phase A22**:** it represents the experimental stage, in fact the users have to evaluate the system progressively while executing the tasks in terms of the already defined criteria. Then he/she allocates to every criterion a mark according to his/her level of satisfaction and according to a well predefined scale.
- Phase A3: it represents the analysis phase. In this level, the evaluator calculates to every user the level of satisfaction N_s that translates the level of personalisation N_p of the system applying the following formula:

$$N_s = N_p = \frac{\sum_{K=1}^{n} W_K \times I_K \times N_K}{\sum_{k=1}^{n} W_K \times I_K}$$

With
n: number of criteria the user is concerned with.
W_K: the weight of interest representing the importance the user gives to this criterion.
I_K: the criterion index of activation.
1 if the user U is concerned with the criterion k
0 if not
N_K: the mark attributed by the user to the criterion k
0 if the user is not satisfied at all
0.25 if the user is a bit satisfied
0.5 if the user is fairly satisfied
0.75 if the user is satisfied
1 if the user is very satisfied

The result is a value restricted between 0 and 1.0 so that the more the satisfaction level is close to 1 the more the system is adapted to the user. To judge the system according to its degree of personalization, we calculate the average of the satisfaction level N_s of all the subject participated in the evaluation.

IV. Case Study: in ITS Domain

● Context: Our work is a part of the project « MOUVER.PERSO» achieved with the collaboration of National Institute of Research on Transport and its Security (INRETS). This system aims to incite the usage of collective transport by ensuring the complementarity between different modes of transport and ameliorating the quality as well as the availability of the personalized information.

● Protocol test: The objective of this experience was to evaluate the demonstrator MouverPerso taking as basis the proposed approach and the criteria we have just defined for the evaluation of SIP.
- The participants: Twenty three people have participated in this evaluation, two expert evaluators, twenty subjects (twenty students in computer science) and a technician.
- The evaluated task: We focused on a representative task of the application, a research of itinerary after adding an appointment. This task needs the consultation of at least three interfaces of the application (an interface which permits to add an appointment, an interface which permits to consult the itinerary details and an interface which permits to consult the appointment list). The dynamic task is represented by Statechart diagram from UML (see fig. 2).
- Tools and techniques: In this experimentation, several tools were used namely:
▪ The general questionnaire: the evaluator invites the user to fill in this questionnaire which includes some general information such as last name, first name, email, age, gender. This questionnaire is distributed among the subjects in the pre-experimental step.
▪ The criterion index-card: the user makes use of his/her index-card to judge the system for example s/he could attribute weights and marks, mention problems and/or draw

the attention to them and may propose ideas to improve the system during the experience.

▪ A PDA (Portable Digital Assistant): is a personal assistant taking the form of a mobile digital appliance. It consists of a computer equipped with a tactile screen and a stylet (fig. 6).

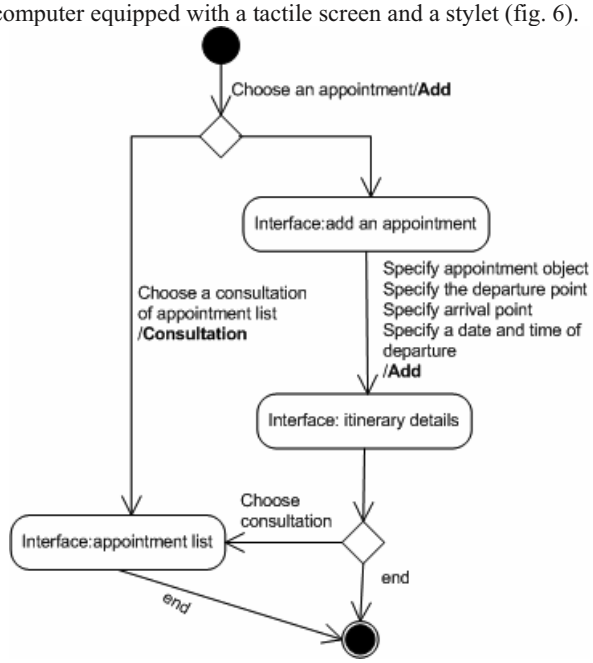


Fig. 2. The dynamic task of adding an appointment.

V. Results and Analysis

This part represents the principal results issued from the test and it is concerned with the dimensions related to the content and container personalization.

A. Content Personalization

Three criteria were considered in the personalization of content: the preferences, point of focus and user experience as we see in fig.3.: 47% of the users are very satisfied, 41% are satisfied, 6% are less satisfied and finally 6% of them are not satisfied at all (see fig. 3).

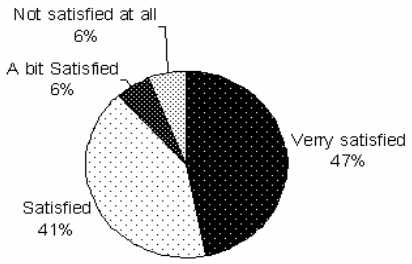


Fig. 3. The subjects' opinion about the personalization related to the content.

In order to give more details about the students' opinion concerning the personalization content, we have compared the average of their level of satisfaction according to the three criteria related to the quality of content personalization. For preferences and experiences, the averages of satisfaction in connection with these criteria exceed 0.6. For the interests the average is less satisfying (see fig. 4).

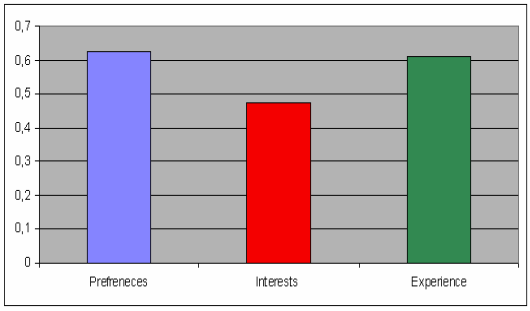


Fig.4. Average of the student's level of satisfaction per personalized criterion related to the content.

B. Container Personalization

The personalization related to the container groups four criteria: adaptation to interactive platform, adaptation to environment, adaptation to user's behaviour and finally adaptation to users' physical capacities (accessibility). The rate of satisfaction is 33% for the very satisfied subject, 17% for the satisfied and 50% for the bit satisfied ones (see fig. 5).

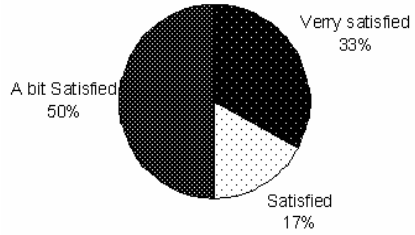


Fig. 5. The subjects' opinion about the personalization related to the container.

In order to give more details about the students' opinion concerning the personalization related to the container, we have compared the average of their level of satisfaction according to the four criteria related to these dimensions.

Fig.6. Test with PDA

After having tested the application in using two different interactive supports (PC, PDA) (see fig.6.) the subject gives his/her opinion about the adaptation of the system to the interactive platform. Concerning the adaptation criterion of the users behaviour, the students' opinion are centred on the neutral response (the average = 5), while as showed in fig.7,

the student have a disapproving opinion about the adaptation to the users' physical capacity and to the environment (the average <4).

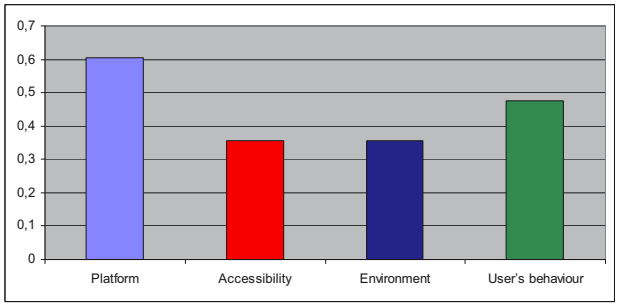


Fig.7. Average of the student's level of satisfaction per personalized orientation related to the container.

VI. Problems Mentioned by the Subjects

The fig. 8 shows that the majority of the subjects have mentioned some problems related to experience, accessibility and users behaviour. About accessibility a problem is cited several times which is the smallness of the characters that causes a bad legibility. Concerning the behaviour, the subjects notice the lack of warning signals during the validation task. The subjects had mentioned few problems related to the preferences, to interests and the interactive platform. This could be explained by the importance the designers of this system had given to the dimensions related to the personalization of the content and to the adaptation to the interactive platform when compared with the others criteria.

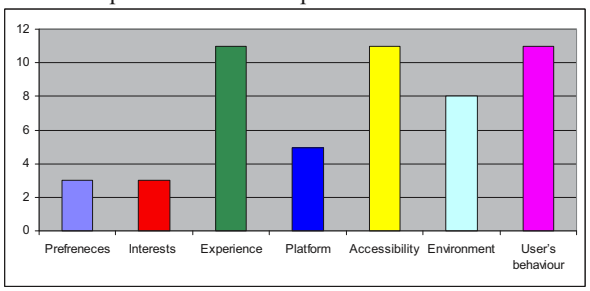


Fig.8. Problems repartition according the evaluation criteria evaluation.

VII. Conclusion and Perspectives

In this article we have mentioned the insufficiencies in the evaluation of PIS and proposed a method to evaluate such systems. This method was tested among a group of subjects using the demonstrator MouverPerso. The information we have collected allowed us to know the subjects' level of satisfaction, the problems they have encountered and the ideas they proposed to ameliorate this system. This method is based on the explicit user intervention and on the existence of a real system (model, prototype or final system). Despite that it provides concise results, since they are explicitly obtained from users' opinions, the evaluator and user spend much time to achieve PIS evaluation. It also disturbs the user in his/her main activity. Besides, this evaluation needs several iterations to find maximum of problems. These reasons make us think about another evaluation method that does not need the direct intervention of the user. The principle of this proposed method which makes the object of our future research will be based on the usage traces of the system.

Acknowledgements

The present research work has been partially supported by the "Ministère de l'Education Nationale, de la Recherche et de la Technologie», the «Région Nord Pas-de-Calais» and the FEDER (projects MIAOU, EUCUE, SART), l'ANR ADEME (Viatic.Mobilité), la PREDIM (MouverPerso). The authors gratefully acknowledge the support of these institutions.

References

[1] R. Barrett, P. Maglio and D. Kellem, How to personalize the Web, in Proceedings of CHI'97, 1997.

[2] I. Cingil, A. Dogac and A. Azgin, A broader approach to personalization, Communications of the ACM 43 (8) 136–141, 2000.

[3] P. Maglio and R. Barrett, Intermediaries personalize information streams, Communications of the ACM 43(8) 96–101, 2000.

[4] C. Basu, H. Hirsh, and Cohen, W., Recommendation as Classification: Using Social and Content-Based Information in Recommendation. In C. Rich and J. Mostow (Eds.), Proceedings of the 15th National Conference on Artificial Intelligence, pp. 714-720. Madison, USA, AAAI Press/MIT Press, 1998.

[5] R. Hoyer, and O.Czogolla, Approach to personalised information services to public transport. In 9th world congress on intelligent transportation systems, Chicago Illinois, October-14-18, 2002.

[6] G.D Abowd, A.K.Dey, P.J. Brown, M. Smith, and P. Steggles, Towards a Better Understanding of Context and Context-Awareness. Lecture Note In Computer Science, Vol. 1707. Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, Karlsruhe, Germany, pp. 304-307, September, 1999.

[7] R.Wilson., E.N. Corlett, Evaluation of human work, a practical ergonomics methodology.Taylor&Francis, 1990.

[8] B. Senach, Evaluation de l'ergonomie des IHM, Actes du Congrès ERGO-IA'90, ergonomie et informatique avancée, Biarritz, 1990

[9] A. Whitefield and al., A framework for human factors evaluation. Behaviour & Information Technology, vol 10, n°1, pp. 65-79, 1991.

[10] M. Sweeney and al., Evaluating user-computer interaction: a framework. Int. J. Man-Machine Studies, 38, 689-711, 1993.

[11] S. Balbo, Evaluation ergonomique des interfaces utilisateur, un pas vers l'automatisation. Thèse de doctorat, Université de Grenoble I, 1994.

[12] J-C. Tarby, H. Ezzedine, , C-D Tran,., P. Laporte, and C. Kolski, Traces using aspect oriented programming and interactive agent-based architecture for early usability evaluation: basic principles and comparison. J. Jacko (Ed.), Human-Computer Interaction, Part I, HCII 2007, LNCS, Springer-Verlag, pp. 632-641, 2007.

[13] J. Huart, C. Kolski, M. Sagar, Evaluation of multimedia applications using inspection methods: The Cognitive Walkthrough case. Interacting With Computers,16 ,pp. 183-215. 2004.

[14] H. Ezzedine, A.Trabelsi, C. Kolski Modelling of an interactive system with an agent-based architecture using Petri nets, application of the method to the supervision of a transport system. Mathematics and Computers in Simulation, 70, pp. 358-376. 2006.

[15] M. Soui, M. Abed, C. Kolski, K. Ghedira, Criteria devoted to evaluate personalized interactive systems. M. Abid, A. Hadj Kacem, M. Jmaiel, M. Lahiani (Ed.), Nouvelles tendances technologiques en génie électrique & informatique, GEI'2007, Edition CPU, Sfax, pp. 119-125, mars, ISBN 978-9973-61-631-9 , 2007.

Dynamis: Dynamic Overlay Service Composition for Distributed Stream Processing

Farshad A. Samimi and Philip K. McKinley
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48823, USA
farshad@ieee.org, mckinley@cse.msu.edu

Abstract— **This paper addresses the problem of mapping software services onto an overlay network, specifically, the probing to locate suitable nodes on which to instantiate or configure data processing operators. We propose a distributed algorithm, called Dynamis, that can improve existing probing algorithms. Experimental results on the PlanetLab testbed show that Dynamis can dramatically reduce probing overhead while producing high-quality services.**

I. Introduction

Processing data streams as they are delivered across a network is essential to many applications. In the case of data gleaned from sensor networks, for example, processing the data as it is collected supports real-time applications and facilitates later searching and analysis of data repositories. While the components of some data processing services are relatively static, others depend on dynamics in the user population and their queries, in which case the services need to be dynamically composed and reconfigured as conditions change. Realizing this functionality has been facilitated by the advent of *overlay networks*, in which end hosts form a virtual network atop a physical network. The presence of *hosts* along the paths between endpoints enables intermediate processing of data streams, without modifying the underlying network protocols or router software.

In this paper, we focus on the problem of mapping distributed services onto an overlay network. This functionality is an integral part of any overlay-based streaming framework and typically requires a probing mechanism to locate suitable nodes on which to instantiate new data processing operators [1], or to reconfigure and possibly share existing operators [2]. The probing protocol should incur minimal traffic overhead while producing a high-quality mapping of services onto the overlay infrastructure. The quality can be measured in terms of metrics such as end-to-end delay, load balance, security, and cost.

The contributions of this study are threefold. First, we propose *distributed selection*, an optimization technique that supports the design of efficient probing mechanisms. We demonstrate that applying distributed selection to probing algorithms can significantly reduce probing overhead. Second, we introduce an extensible algorithm based on distributed selection, called Dynamis, to realize efficient probing for overlay service composition. Third, we report results of an experimental study on the PlanetLab Internet testbed, where we assess the performance of Dynamis and other service composition algorithms.

The remainder of this paper is organized as follows. Section II provides background and discusses related work. Section III formulates the problem of service composition and probing. Section IV introduces Dynamis, and Section V describes the experimental investigation. Finally, in Section VI we conclude the paper and discuss future directions.

This work was supported in part by the U.S. Dept. of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, and in part by National Science Foundation grants EIA-0000433, EIA-0130724, and ITR-0313142.

Farshad Samimi is currently with GoldSpot Media, Inc. This research was conducted while he was at Michigan State University.

II. Background and Related Work

The service composition problem arises in distributed environments where the system needs to set up and bind a number of entities in order to realize services [3]. With the emergence of overlay networks and adaptive middleware technologies [4], dynamic composition of overlay services has recently attracted considerable attention [5]–[7]. Overlay networks provide a chassis on which to deploy services, and adaptive middleware enables dynamic instantiation and configuration of distributed service components. Overlay service composition is particularly useful in distributed processing of data streams [8], [9].

A fundamental issue in overlay service composition is the probing method to locate and select a set of nodes on which to execute stream processing operators (service elements). Researchers have investigated two main aspects of this problem: locating nodes on which to execute the operators [1], [10]–[12] and sharing of services and processed data [2], [13]. In both cases, most prior research has addressed situations where services already exist in the network and need to be connected together to form a suitable *service graph*. Gu et al. [10] introduced SpiderNet, a peer-to-peer service composition framework that performs distributed bounded probing. A key property of the SpiderNet algorithm is that probes are distributed only to nodes capable of executing the required functions. Pietzuch et al. [1] proposed SBON, a protocol that locates suitable nodes on which to place stream operators. The SBON design is based on a "cost space," a multi-dimensional metric where the distance between nodes is an estimate of the cost of routing between them (in terms of desired measurements such as latency and processing power). Liang and Nahrstedt [11] have addressed the problem where data streams from multiple sources are processed and aggregated

to be delivered to multiple destinations. Finally, Repantis et al. [2] proposed the Synergy framework as a means to reuse existing streams and processing components when composing services. These studies and others have significantly advanced the areas of service composition and data stream processing.

Dynamis complements the above approaches by realizing a generic optimization technique based on distributed selection. As we will show, distributed selection can be applied to existing probing protocols, such as those in SpiderNet [10] and SBON [1], in order to reduce probing costs. We implemented and evaluated Dynamis using Service Clouds [14], an overlay-based infrastructure to compose autonomic communication services. A service cloud can be viewed as a collection of hosts whose resources are available to enhance services (e.g., in terms of fault tolerance and quality of service) transparently to the endpoints. Effectively, overlay nodes provide a "blank computational canvas" on which services can be instantiated and reconfigured as needed. Here, we use different probing algorithms in Service Clouds and evalutate the differences in the resulting service mappings.

III. Problem Formulation

In this section, we provide basic definitions and formally state the probing problem for distributed service composition.

Service Element (Operator). A service element $S = \{F, R, I, O\}$ is a service entity executing on a single node, where F specifies the set of functions carried out by the service; R defines the resource requirements of the service, for example, {memory, processing power, output bandwidth}; I specifies acceptable input, for example, {bit rate, resolution} in a video stream; and O states the generated output specifications, for example, {size, fps} in a video stream.

Service Path. A service path P is an alternating directed sequence $P : n_0, l_0, n_1, l_1, ..., n_n, l_n (n > 0)$ of overlay nodes and overlay links l_i, where $l_i = (n_{i-1}, n_i)$, such that each node executes one or more service elements (S_i) each time it is visited on the sequence. [1] Figure 1 depicts an example service path S consisting of three service elements distributed between two endpoints. We note that it is necessary to specify *both* nodes and links in the path, since an overlay network may be multichanneled, with multiple overlay links following different physical paths between the same two nodes [15].

Service Graph. A service graph $\lambda = \cup\{P_i\}$ comprises the union of one or more connected service paths. Figure 1 shows a service graph that forms a multicast tree.

Service Graph Quality. The quality of a service graph is the end-to-end quality observed at the endpoints. The quality measurement is domain specific and may include overlay stretch properties such as end-to-end delay and packet loss, or non-functional aspects such as security, reliability, and cost.

Hosting. We say that a node can host a service element if that node has available resources to execute a service element and satisfy domain-specific criteria of the service graph, such as reliability, security, and end-to-end delay.

[1] In graph theory, such a traversal is called a *walk*, and a *path* is a walk in which no vertex is repeated. Since in the service composition literature the term *service path* is common and used loosely, we also use this term.

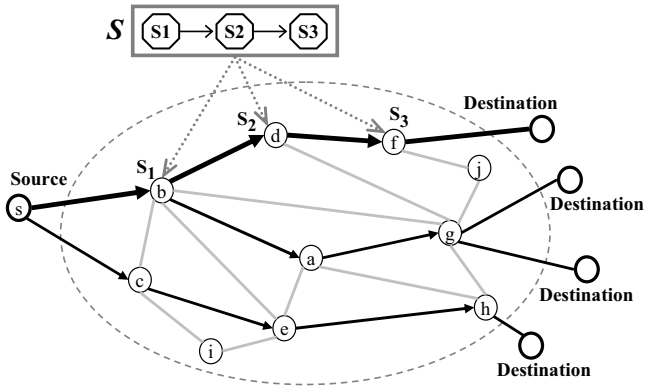


Fig. 1. Service graph example (highlighting a service path).

Comparable Service Graphs. Two service graphs are comparable if they map the same functionality onto the overlay network. In Figure 2, the service paths at the bottom of the figure, $a - g$ and $c - e - h$, both host service elements S_1, S_2, and S_3, so they are comparable; they are not comparable to the $d - f$ graph, which hosts only services S_2 and S_3.

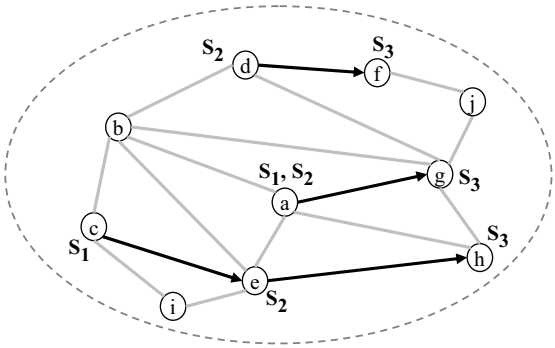


Fig. 2. Examples of service graphs.

This work focuses on composing service *paths*; the general problem of composing service *graphs* is part of our ongoing work. Figure 3 shows examples of two overlay service paths. In these figures, the notation $\{S_i, ..., S_j\}$ represents an unordered set of service elements, that is, functionality of members is commutative. The notation $(S_i, ..., S_j)$ represents an ordered set of service elements, that is, the functionality of each member depends on the previous one and is non-commutative.

Formally, we can state the problem addressed in this paper as follows: Given an overlay graph G of n nodes and a service specification S, find a path P in G such that P can host S.

IV. Dynamis Probing Algorithm

In a probe-based approach to service composition, multiple probes are sent to find service path candidates. A probe contains the requirements of the service path, including the ordering constraints, resource requirements, and expected end-to-end quality. The Dynamis algorithm is based on *distributed selection*, which applies the principle of optimality, namely, that in an optimal sequence of decisions or choices, each subsequence must also be optimal [16]. We observe that service path composition satisfies the principle of optimality.

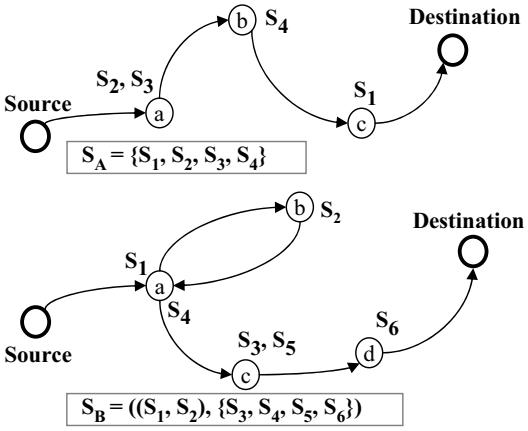


Fig. 3. Service ordering constraints and corresponding service paths.

That is, for any overlay node on an optimal service path, the two partial service paths from that node to the endpoints must also be optimal. We use this observation to design a probing algorithm in which an overlay node drops or forwards a probe based on the quality of the partial service paths found earlier.

Figure 4 depicts the basic operation of the Dynamis probing mechanism, which generalizes an algorithm proposed by Tang and McKinley [15] to construct multipath connections in overlay networks. One of the endpoints initiates probing (path-explore process) by sending the probe for a service path to a subset of its neighbors in the overlay network, according to a predefined branching factor. Thereafter, probes attempt to find their way to the other endpoint. In the algorithm presented here, the destination endpoint (arbitrarily and without loss of generality) starts the path-explore process.

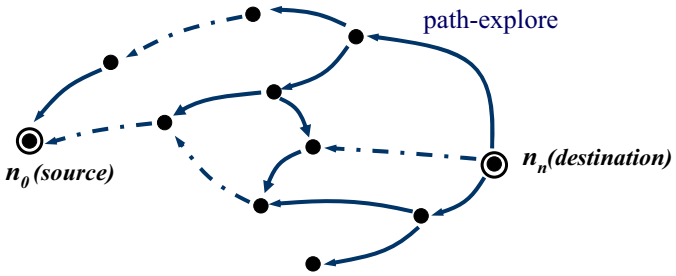


Fig. 4. Basic operation of the composition probing algorithm.

To measure quality of a service path λ, denoted $\psi(\lambda)$, we extend the load balancing metric from Synergy [2] to include the round-trip time as a factor. Specifically,

$$\psi(\lambda) = \omega_p \sum \frac{p_{s_i}}{q_{v_i} + p_{s_i}} + \omega_b \sum \frac{b_{s_i}}{c_{v_i} + b_{s_i}} + \omega_d \frac{D}{D_{max}} n \quad (1)$$

where the terms of the formula are defined in Table I. This metric quantifies the quality of a service path based on the processing and bandwidth load of overlay nodes on the service path, as well as its end-to-end overlay delay. The smaller the $\psi(\lambda)$ value, the better the quality of the service path.

The key property of the Dynamis algorithm is that rather than performing selection only at an endpoint, the selection is distributed. An overlay node forwards a probe only if it describes a partial service path of significantly better quality than the quality of *comparable* service paths described by probes forwarded previously. As we shall demonstrate in Section V, this approach can dramatically reduce the overhead of probing while retaining high quality.

TABLE I

Notation	Meaning
λ	service path
p_{s_i}	processing resource required for service element S_i
q_{v_i}	residual processing capacity on node v_i
b_{s_i}	uplink bandwidth required for service element S_i
c_{v_i}	residual uplink bandwidth on node v_i
D	end-to-end delay of an overlay path
D_{max}	maximum acceptable end-to-end delay
n	number of nodes in the service path
$\omega_p, \omega_b, \omega_d$	experimental cofactors

Algorithm Sketch. Upon receiving a path-explore probe, each node n_i inspects the probe and performs one of the following actions (details can be found in [17]):

(i) Node n_i drops the probe, in any of the following conditions:

(a) The service path violates the expected quality (e.g., end-to-end delay) at this point.

(b) Node n_i cannot satisfy resource requirements for hosting at least one of the service elements that can be added to the partial service path explored to this point (hosting).

(c) The quality of the partial service path explored so far is not *significantly* better (e.g., by at least 5%—as specified in the configuration) than the quality of the best *comparable* partial service path described by a probe already forwarded by node n_i.

(d) The quality of the partial service path explored so far is not better than the quality of a comparable partial service path described by a probe currently buffered to be forwarded.

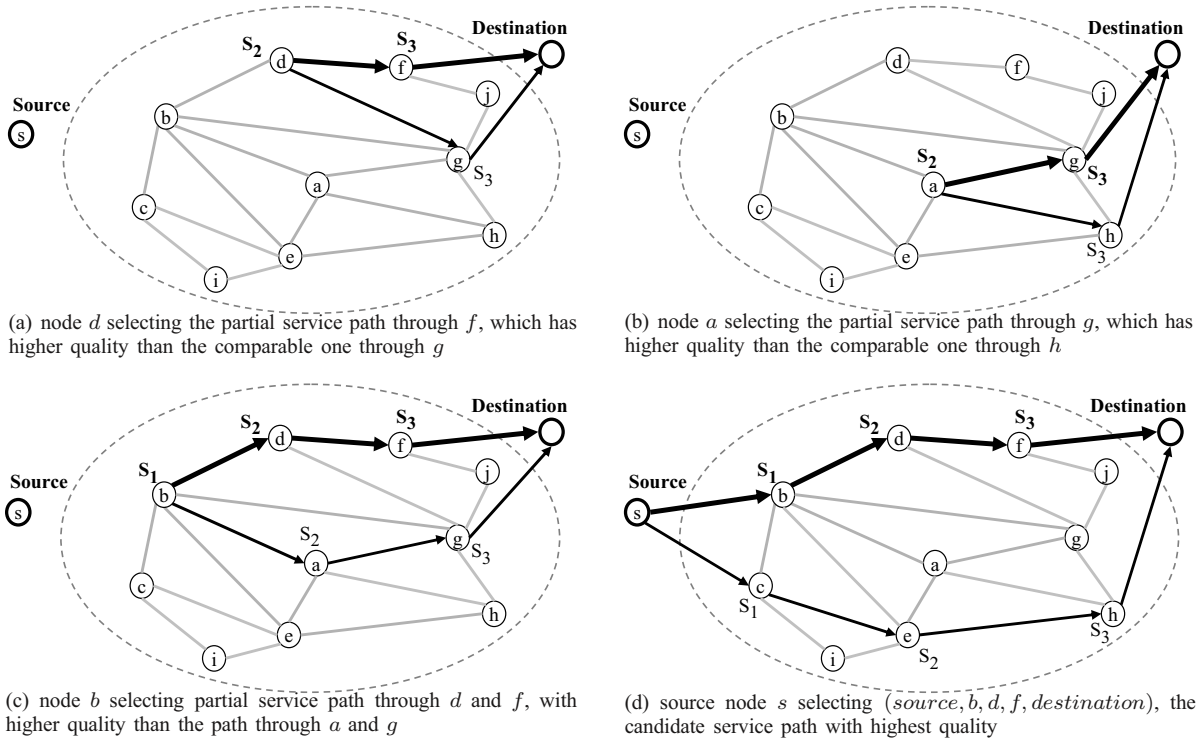
(ii) Otherwise, node n_i:

(a) Updates the probe, adding itself to the partial service path described by the probe.

(b) If the partial service path achieves the requested service path, node n_i announces a candidate service path. If node n_i is the target endpoint of the probe, then this announcement is local; otherwise, node n_i forwards the probe to the target endpoint.

(c) Otherwise, node n_i buffers the probe, replacing any probe in the buffer that describes a comparable partial service path. Node n_i periodically forwards all probes in the buffer; this period is called an *epoch*. Node n_i forwards a probe to a subset of "qualified" neighbors. Criteria for qualification include trust relationship, cost, and availability of a particular functionality.

Example. Figure 5 demonstrates a simplified run-time operation of the Dynamis algorithm, in which the service path $S = (S_1, S_2, S_3)$ is being mapped to overlay nodes. In Figure 5(a) two comparable partial service paths have been found up to node d during a distributed selection epoch. The algorithm forwards only the probe describing the path of highest quality, and then only if the quality is significantly better than the best comparable service path forwarded in



(a) node d selecting the partial service path through f, which has higher quality than the comparable one through g

(b) node a selecting the partial service path through g, which has higher quality than the comparable one through h

(c) node b selecting partial service path through d and f, with higher quality than the path through a and g

(d) source node s selecting $(source, b, d, f, destination)$, the candidate service path with highest quality

Fig. 5. Example run of the Dynamis probing algorithm: mapping $S = (S_1, S_2, S_3)$ to the overlay network.

a previous epoch. Let us assume that S_2 can be hosted by node d, so node d forwards $(d, f, destination)$. In a similar way, in Figure 5(b), $(a, g, destination)$ is selected. Then, in Figure 5(c) node b receives two comparable partial service paths from nodes d and a. Again, applying distributed selection (and assuming S_1 can be hosted by node b), only the partial path with higher quality is selected if both of the probes are received in the same epoch. If the probes are received during different epochs, then the one received later is selected only if it has significantly better quality. Now, let us assume that only $(b, d, f, destination)$ is selected, which is forwarded to the source node as a candidate service path. Eventually, the source node selects the candidate with the highest quality, $(source, b, d, f, destination)$ in this example, and maps the service elements onto the corresponding overlay nodes. In approaches that do not use distributed selection, nodes typically forward all of the probes which they receive.

Parameters. Table II gives the parameters used in the algorithm. T is the buffer time-out period, or *epoch* duration; when the buffer is empty and a probe is placed in it, a timer starts that flushes the buffer after T milliseconds. $\overline{B}$, $\overline{H}$, and $\overline{W}$ are branching factor parameters that control the overhead of the probing in three respective ways: budgeted, limited-hop, and bounded. In *budgeted* forwarding, each node forwards a probe to $\overline{B}$ qualified neighbors selected at random. In limited-hop forwarding, a probe is discarded if it has not found a service path after traversing $\overline{H}$ overlay nodes. In *bounded* forwarding, each node forwards at most $\overline{W}$ probes for a service composition session. Finally, $\underline{Q}$ specifies the minimum service path quality improvement expected, relative to the quality of a comparable service path in a probe forwarded previously, in order to forward a probe in question.

TABLE II

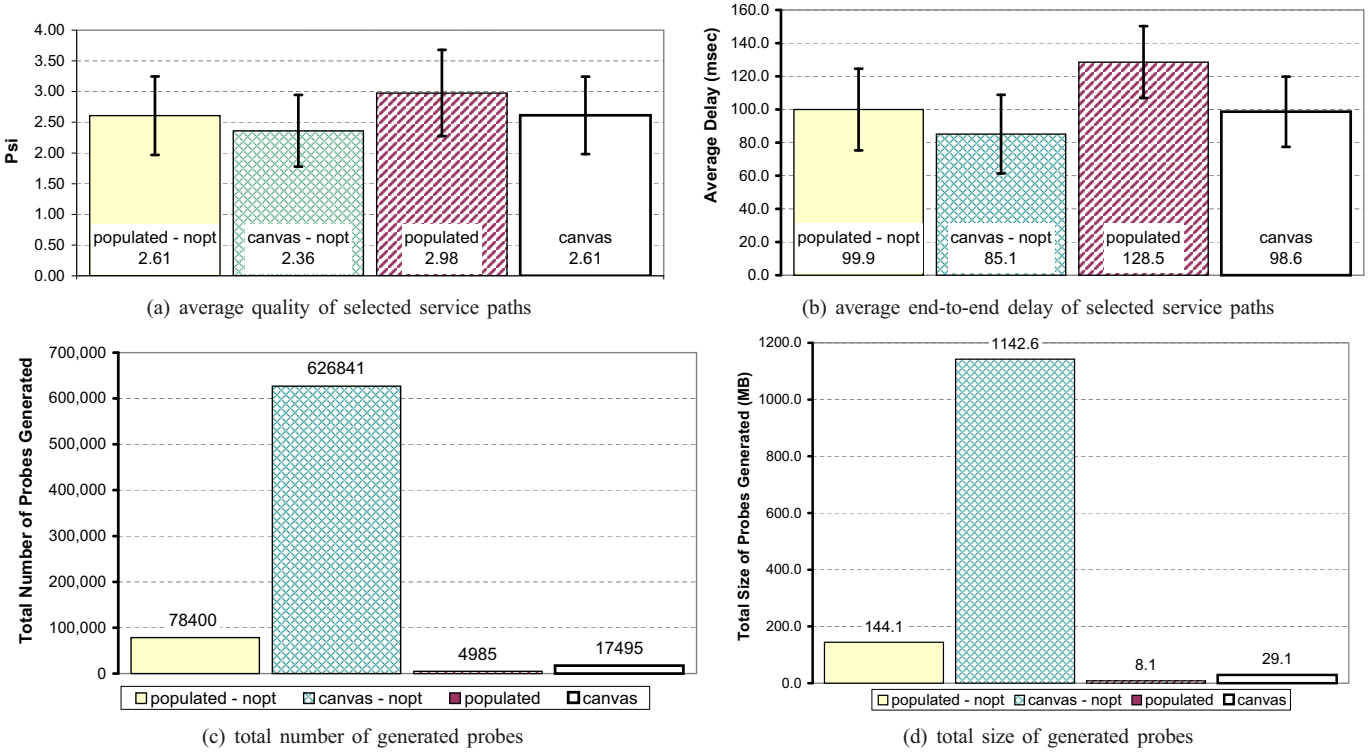
Notation	Meaning
T	buffer time-out
$\overline{H}$ upper bound	number of hops a probe can traverse
$\overline{W}$ upper bound	number of probes forwarded at each node
$\overline{B}$ upper bound	number of forwards for a probe at each node
$\underline{Q}$	expected quality improvement

V. Experimental Evaluation

We assess the performance of Dynamis in composing services on PlanetLab [18], an Internet research testbed comprising hundreds of Linux-based nodes distributed throughout the world. We have used the Service Clouds [14] prototype to apply Dynamis to different probing strategies. In these experiments we assume all service elements need to be executed in order (ordered service path). Due to space limitations, many experimental results are omitted here, but can be found in [17].

Test Setup and Procedure. Considering establishment of service paths between two nodes on the Internet, we first show that the proposed approach significantly reduces probing overhead by incorporating distributed selection (rather than selection at an endpoint), while still finding high-quality service paths. We use Formula 1 (Section IV) to measure the quality of a service path. Next, we evaluate the quality of the selected service paths in different approaches and configurations in terms of end-to-end delay. In particular, we assess the effect of $\underline{Q}$ and T parameters.

We measure the quality of the best service path found and the corresponding probing overhead. This implementation realizes a simplified Dynamis algorithm, which assumes two service elements of the same service path do not execute on the same overlay node (hence, probes are not forwarded to nodes



(a) average quality of selected service paths

(b) average end-to-end delay of selected service paths

(c) total number of generated probes

(d) total size of generated probes

Fig. 6. The quality and the end-to-end delay of selected services path vs. overhead of probing in the four different strategies.

they have already traversed). The test procedure cycles through source-destination pairs of nodes, submitting service requests to the system and allocating resources. We have implemented a simple protocol to reserve virtual resources. The source node performs the final service path selection and sends messages to reserve the required amount of resources on the overlay nodes along the selected service path.

Table III shows the settings of parameters. In case of multiple values for a parameter, the one in parentheses is used by default; unless stated otherwise. In these tests, all service specifications contain four ordered service elements, which consume the same amount of resources: 20 units of CPU processing power and 200Kbps of bandwidth. We conducted the experiments on 28 PlanetLab nodes, each assumed to have 100 units of CPU processing power and 1024Kbps uplink bandwidth. The results presented are the average of five samples of service composition between each pair of nodes.

TABLE III

Parameter	Value
distributed service selection	(enabled) , disabled
initial resource loads (CPU and bandwidth)	from (0%) to 60%
T	(500) , 1500 , 2000 msec
$\underline{Q}$	(0%) , 2% , 3% , 5% , 10%
$\overline{H}$ upper bound	N/A
$\overline{W}$ upper bound	(unlimited), 3000
$\overline{B}$ upper bound	unlimited
p_{s_i}	20 units of normalized CPU time
b_{s_j}	200 Kbps
D_{max}	300 msec
$\omega_p, \omega_b, \omega_d$ cofactors	1.0

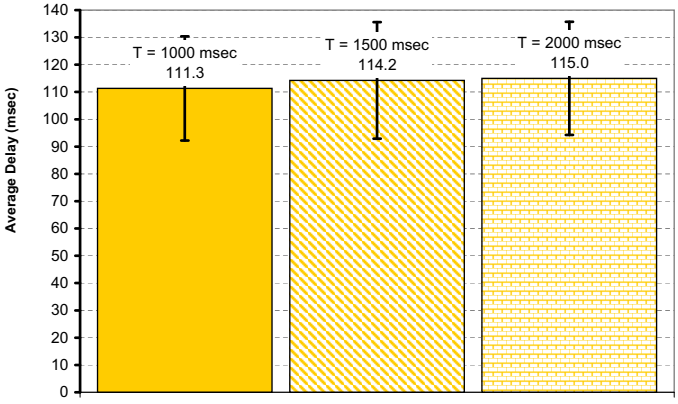
Results of Experiments. These tests compare four major strategies:

- *populated - nopt:* composes a service path using nodes that already host the required services; does not use distributed selection.
- *canvas - nopt:* considers the overlay as a blank computational canvas, so any service can be mapped to any node with sufficient resources; does not use distributed selection.
- *populated:* composes a service path using nodes that already host required services; uses distributed selection.
- *canvas:* considers the overlay as a blank computational canvas and uses distributed selection.

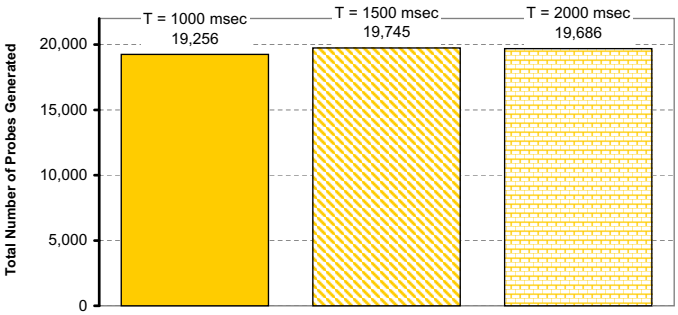
Figures 6(a) and 6(b) show the average quality and end-to-end delay for each of the strategies (95% confidence intervals included). While the values for the distributed selection cases ("populated" and "canvas") are slightly higher (therefore worse) than those for the first two cases, the differences are not significant. On the other hand, Figures 6(c) and 6(d) show that the probing overhead is dramatically reduced by distributed selection (over 93% in the " populated" strategy, and over 97% in the "canvas" strategy).

Figure 7 shows the effect of the parameter T in "canvas" strategy. These plots show no significant change as T is set to the different values. This observation is expected, since each node compares the quality of a partial service path both to all other ones received during the same epoch, and to those forwarded in previous epochs. Thus, unless the value of T is so small that few probes are received during an epoch, increasing the buffer timeout does not change the behavior of the strategies. Also, results from additional tests show that increasing $\underline{Q}$ has little effect on overhead, but does

increase end-to-end delay. We can conclude an epoch duration of $500msec$ is sufficient to significantly reduce the probing overhead of composing high-quality service paths, without the need for $\underline{Q}$, at least within the realm of these experiments.



(a) average delay of selected service paths



(b) total number of generated probes

Fig. 7. Effect of T on the service path delay and probing overhead ("canvas" strategy with Q = 5%).

VI. Conclusions and Future Work

We described Dynamis, a probing algorithm to support composition of distributed overlay services. Dynamis is based on distributed service selection, which reduces the overhead of probing to locate suitable nodes on which to instantiate services. We presented the Dynamis algorithm and used it to empirically assess performance of different service composition strategies on the Internet. The experimental results show that using distributed selection reduces the probing overhead in service composition, while still finding high-quality service paths. Future work can include experiments with fewer assumptions, such as evaluation within a heterogeneous assortment of nodes with different capabilities and resources; as well as addressing quality of service in terms of non-functional requirements, such as trustworthiness and reliability. Furthermore, our ongoing research addresses the design of an autonomic framework to compose adaptive distributed stream processing services, including real-time data streams generated by sensor networks that monitor ecosystems.

Further Information. Related publications and software on the RAPIDware project can be found at the following website: `http://www.cse.msu.edu/rapidware`.

References

[1] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, (Washington, DC, USA), pp. 49–60, IEEE Computer Society, 2006.

[2] T. Repantis, X. Gu, and V. Kalogeraki, "Synergy: Sharing-aware component composition for distributed stream processing systems," in *Proceedings of the 7th ACM/IFIP/USENIX International Middleware Conference (MIDDLEWARE 2006)*, vol. 4290 of *LNCS*, (Melbourne, Australia), Springer-Verlag, November 2006.

[3] S. D. Gribble, M. Welsh, J. R. von Behren, E. A. Brewer, D. E. Culler, N. Borisov, S. E. Czerwinski, R. Gummadi, J. R. Hill, A. D. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Y. Zhao, "The Ninja architecture for robust Internet-scale systems and services," *Computer Networks*, vol. 35, no. 4, pp. 473–497, 2001.

[4] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing adaptive software," *IEEE Computer*, vol. 37, no. 7, pp. 56–64, 2004.

[5] P. Grace, G. Coulson, G. Blair, L. Mathy, D. Duce, C. Cooper, W. K. Yeung, and W. Cai, "GRIDKIT: Pluggable overlay networks for Grid computing," in *Proceedings of International Symposium on Distributed Objects and Applications (DOA)*, (Larnaca, Cyprus), pp. 1463–1481, October 2004.

[6] J. Xiao and R. Boutaba, "QoS-aware service composition and adaptation in autonomic communication," *IEEE Journal on Selected Areas in Communications*, vol. 23, pp. 2344–2360, December 2005.

[7] X. Fu and V. Karamcheti, "Automatic creation and reconfiguration of network-aware service access paths.," *Computer Communications*, vol. 28, no. 6, pp. 591–608, 2005.

[8] V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan, "Resource-aware distributed stream management using dynamic overlays," in *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS 2005)*, (Columbus, OH, USA), pp. 783–792, IEEE Computer Society, June 2005.

[9] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The Design of the Borealis Stream Processing Engine," in *2nd Biennial Conference on Innovative Data Systems Research (CIDR'05)*, (Asilomar, CA, USA), pp. 277–289, January 2005.

[10] X. Gu, K. Nahrstedt, and B. Yu, "SpiderNet: An integrated peer-to-peer service composition framework," in *Proceedings of IEEE International Symposium on High-Performance Distributed Computing (HPDC-13)*, (Honolulu, Hawaii), pp. 110–119, June 2004.

[11] J. Liang and K. Nahrstedt, "Service composition for advanced multimedia applications," in *Proceedings of 12th Annual Multimedia Computing and Networking (MMCN'05)*, vol. 5680, (San Jose, California), pp. 228–240, January 2005.

[12] B. J. Bonfils and P. Bonnet, "Adaptive and decentralized operator placement for in-network query processing.," *Telecommunication Systems*, vol. 26, no. 2-4, pp. 389–409, 2004.

[13] S. Seshadri, V. Kumar, and B. F. Cooper, "Optimizing multiple queries in distributed data stream systems," in *Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW)*, (Atlanta, GA, USA), pp. 25–30, IEEE Computer Society, April 2006.

[14] F. A. Samimi, P. K. McKinley, S. M. Sadjadi, C. Tang, J. K. Shapiro, and Z. Zhou, "Service Clouds: Distributed infrastructure for adaptive communication services," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 4, pp. 84–95, September 2007.

[15] C. Tang and P. K. McKinley, "Improving mutipath reliability in topology-aware overlay networks," in *Proceedings of the Fourth International Workshop on Assurance in Distributed Systems and Networks (ADSN), held in conjunction with the 25th IEEE International Conference on Distributed Computing Systems*, (Columbus, Ohio), June 2005.

[16] G. Brassard and P. Bratley, *Fundamentals of Algorithmics*. Prentice Hall, 1996.

[17] F. A. Samimi and P. K. McKinley, "Dynamis: Dynamic overlay service composition for distributed stream processing," Tech. Rep. MSU-CSE-06-39, Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, December 2006. Available at `http://www.cse.msu.edu/~farshad/serviceclouds`.

[18] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," in *Proceedings of ACM Workshop on Hot Topics in Networks*, (Princeton, New Jersey), pp. 59–64, October 2002. `http://www.planet-lab.org/`.

Wings4Symbian: A Pervasive Computing Middleware for Symbian OS Mobile Devices

Olympio C. Silva Filho, Danilo F. S. Santos, Angelo Perkusich, Emerson Loureiro and Hyggo Almeida

Abstract— **In this paper we present the main architectural aspects about the implementation of a pervasive middleware (named Wings4Symbian) used to develop pervasive applications for smartphones running the Symbian Operating Systems. Smartphones play an important role on providing users with access to services dispersed on the environment. In terms of software infrastructure, the Symbian Operating System is a reliable platform for the realization of pervasive computing vision. Key factors such as modularization, multi-tasking, real-time kernel, robustness, among others, are strongly supported. The introduced middleware provides mechanisms for service provision, host discovery, and context awareness. One application developed on top of the *Wings4Symbian* middleware is also discussed, focusing on its reconfiguration features.**

Index Terms— **Middleware, Symbian Smartphones, Pervasive Computing**

I. Introduction

Pervasive computing [1] has emerged as a prominent research field in the last years. In the context of pervasive, computing and communication capabilities will be available not only in traditional computing appliances, such as personal computers, personal digital assistants (PDAs), and cellular phones, but also in everyday objects like televisions, refrigerators, cars, air-conditioners, among others [2]. Moreover, all these computing devices will be transparently integrated into our lives, with the goal of providing us with relevant information and services.

Smartphones play an important role in the context of pervasive computing [1] mainly due to three reasons [3], [4], [5]. Firstly, applications can be easily deployed on them and can use their connectivity to communicate with the environment. Secondly, smartphones are very personal items and, thus, remain with their owners most of the time. And finally, they can acquire relevant information about the environment based on different kinds of devices, like sensors, cameras, and microphones, as well as they can detect available services that are around. Altogether, these features make smartphones the ideal solution for user interaction in pervasive environments.

The Symbian Operating System (Symbian OS) [6], [7], [8] can be considered a reliable platform for the deployment and execution of pervasive computing applications, for the following reasons. Firstly, the Symbian OS is modularized. Therefore, the functionalities of the kernel are provided in building blocks, making the addition of drivers and other pieces of software easy and without major impacts in the overall system. Also, in pervasive computing, system reconfiguration is the norm rather than the exception. Therefore, the modularization feature enables a seamless upgrade of the system. Secondly, Symbian supports multi-tasking, and thus, the kernel is able to switch between different threads. This is important because it allows the execution of multiple pervasive applications concurrently. Third, the Symbian OS has a real-time kernel, which improves the performance of time-sensitive services, which are very important in pervasive environments [9]. Fourth, Symbian is *robust*, in the sense that it is protected against bad written applications, so that they cannot harm other applications running at the same time in the device. All these features, along with an efficient memory and power management system, allow pervasive computing applications to be executed securely, seamlessly, and properly in devices that can go through many hours of operation without recharging batteries.

Based on the introduced scenario and considering the need for an infrastructure to help in the development of pervasive applications for smartphones, the implementation aspects of a pervasive computing middleware for the Symbian operating system, named *Wings4Symbian*, is presented in this paper. The paper is structured as follows. In Section II, the *Wings* middleware architecture is presented. In Section III, the implementation in the Symbian operating system is described. In order to illustrate the application of *Wings4Symbian*, an example scenario and the application development is presented Section IV. Related works are discussed in Section V. In Section VI, the conclusions and the future works are presented.

II. The Wings Pervasive Computing Architecture

The *Wings* architecture is illustrated in Figure 1. It is targeted for pervasive computing systems and is composed of the following modules: *Dynamic Evolution*, *Pervasive Networking*, *Context Awareness*, and *Middleware Facade*. The *Wings* architecture, and more specifically the Pervasive Networking and Context Awareness modules, is a plug-in based approach [10]. The idea is to decouple the functionalities of these modules from the applications, by encapsulating them into plug-ins. Therefore, always when an application is deployed into a given device, the plug-ins for that device are deployed as well, thus enabling the application to run in a

The authors are with the Embedded System and Pervasive Computing Laboratory, Department of Electrical Engineering, Federal University of Campina Grande, C.P. 10105 - 58109-970 - Campina Grande - PB - Brazil, emails: (olympio, danilo, perkusic)@ee.ufcg.edu.br, emerson.loureiro@ucd.ie, hyggo@dsc.ufcg.edu.br

wide range of devices but still keeping its code untouchable. The details of the modules are presented as follows.

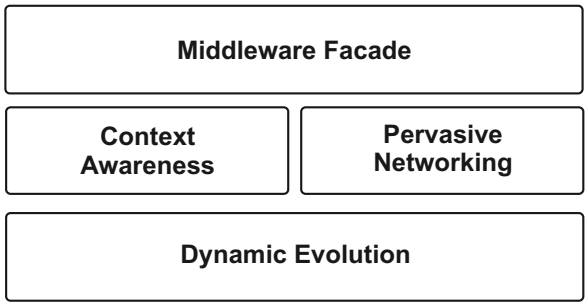


Fig. 1. The *Wings* pervasive computing architecture.

The *Dynamic Evolution Module* allows *Wings* middleware to be updated at runtime. More specifically, it allows the insertion and removal of *Wings*-based plug-ins, without stopping and restarting neither the middleware nor the applications that are being executed. More precisely, new modules can be deployed or removed whenever the user comes/leaves an environment, but the device does not need to be restarted due to this.

The *Pervasive Networking Module* implements two functionalities: *host discovery* and *service provision*. The *Wings* architecture defines two types of plug-ins for implementing these functionalities, namely, *Host Discovery Plug-ins* (HDPs) and *Service Provision Plug-ins* (SPPs). Furthermore, since it is possible to deploy different implementations of these plug-ins, at the same time, both, the host discovery and the service provision, can be performed over heterogeneous protocols. For example, both operations can be executed over UPnP, JXTA, and Bluetooth based networks, thus increasing the number of services and hosts that a device can access.

The *Context Awareness Module* implements mechanisms for enabling applications to retrieve context information, which can be performed either by using *key-value pairs* or *context events*. For the former, each context information is associated with a key, which is used to retrieve the current information. Context events, on the other hand, enable applications to be aware of changes in the context through event notification. In the *Wings* pervasive computing architecture, context awareness is provided by a type of plug-in named *Context Awareness Plug-in*, or *CAP*. Given a specific application domain (e.g., smart homes), the basic idea is that each *CAP* is associated with the domain, thus providing only the context information associated with that given domain.

The *Middleware Facade Module* provides a single interface to access services provided by the available plug-ins in the underlying modules, i.e. the Service Discovery and Context Awareness modules. The need for this module relies on the fact that, when different plug-ins are installed in the middleware, it would be too complicated, from the developer point of view, for applications to access the services provided by these plug-ins directly. All applications would need to be aware of each inserted and removed plug-in, for enabling them to use the services of the recently inserted ones as well as to avoid using the services of the already removed plug-ins. The Middleware Facade module, thus, abstracts all these details from the applications. Besides, this module is also responsible for loading and unloading the plug-ins.

III. The *Wings4Symbian* Pervasive Computing Middleware

The Wings4Symbian middleware is implemented in C++ for the Symbian OS and the implementation is divided into two major parts: the plug-ins and the facade implementation. The former consists of the implementation of the Pervasive Networking and Context Awareness plug-ins (i.e. SPPs, HDPs, and CAPs). Such plug-ins, implemented as polymorphic dynamic linked libraries (DLL's), are responsible for providing the functionalities of the *Wings4Symbian* middleware. As discussed in Section II, the plug-ins can be loaded at runtime so that the middleware can be dynamically extended to satisfy specific needs of applications – context information, service provision, and host discovery. The facade is implemented as shared DLL.

Two frameworks of the Symbian OS were widely used throughout the implementation of the *Wings4Symbian* middleware, namely: the Active Objects Framework and the Plug-in Framework. The former was used to implement the plug-ins and the facade and the later was used in the implementation of the Dynamic Evolution Module. In what follows we present more specific details about the plug-ins and facade implementations.

A. Active Objects and Plug-in Framework

The architecture of the Symbian OS is based on a client/server design [6]. The role of the system servers is to encapsulate resources or services to discipline clients access. Also, considering that Symbian OS enables asynchronous service calls, clients can register to be notified of certain server events. Therefore, event-driven coding is extensively applied through the use of Active Objects (AO). Unlike multi-threading, active objects are a lightweight mechanism to perform event-driven multitasking without decreasing responsiveness, while still saving power and memory resources. An Active Scheduler for each thread waits for events completion issued by any AO of the corresponding thread and executes the code of the appropriate Active Object. The Active Scheduler is non-preemptive, thus the Active Object Framework is a model for scheduling tasks non-preemptively inside one thread.

The Symbian OS framework used to support plug-ins is named ECOM [6], which is an object-factory framework (or object model). It is a generic and extensible plug-in framework for registering and discovering abstract interfaces implementations. They can be loaded and unloaded by the framework at run-time, thus only the implementations required by an application need to be presented. The ECOM framework provides the following elements: an *abstract interface*, which

defines a plug-in type as well as the service it provides; an *implementation of this interface*, that is, the plug-in itself; and a framework for providing clients with access to specific plug-ins. The abstract interface has two types of methods: *abstract methods* and *factory methods*. Abstract methods define functionalities that must be implemented by the plug-in. The factory methods, on the other hand, are used for instantiating a plug-in, by sending requests to the ECOM server.

B. Service Discovery Plug-in

The Service Provision Plug-in, which is part of the Pervasive Networking Module, enables devices to find, advertise, and use services available in a pervasive environment. The most important classes of this plug-in are illustrated in the the class diagram shown in Figure 2 and described as follows:

- *CSpp*: this is the abstract interface of the Service Provision Plug-in. It has four abstract methods *DiscoverServicesL*, *AdvertiseServiceL*, *UnadvertiseService*, and *StopDiscovery*, which specify the plug-in functionalities, and one factory method, *NewL*, which requests the ECOM server to load the needed plug-in implementation into the middleware. The *DiscoverServicesL* is used for searching available services in the environment. It returns a search identifier so that applications are able to stop the search. The *AdvertiseServiceL* method is used for advertising a local service to other devices. Unadivertising a service is made through the *UnadvertiseService* method, which receives the service to be unadvertised. Finally, for stopping a search for services, the *StopDiscovery* method must be used, providing it with a specific search identifier. This is necessary because one client can make parallel searches for services, and thus, such an identifier enables the SPP to stop the correct search.
- *MService*: this interface represents a *Wings* service, either a remote or a local one. It has four methods, *GetName*, *GetDescription*, *GetParameters*, and *GetReturnType*, which provides basic information about a service (i.e., name, description, parameters, and return type). At last, invoking the service is done through the method *Invoke*. This method receives some parameters to be processed by the service provider and a listener, used to receive the response. The listener is necessary due to the asynchronous nature of this method.
- *MServiceProxy*: this abstract class extends the *MService* class and represents a remote service, that is, those discovered by the SPPs. It is worth mentioning that this class will have to make remote calls to the real service implementation. Therefore, as such calls will be performed according to the protocol associated with the SPP, all methods of this interface are left to be implemented by the subclasses.
- *CLocalService*: this abstract class implements the *MService* interface and represents a local service. All methods of *MService* are implemented by this class, except *Invoke*, because it is specific to each service, and consequently must be left to be implemented by its subclasses.
- *MServiceDiscoveryListener*: this interface must be implemented by objects interested in receiving information about the service discovery process. It has two methods: *ServiceDiscovered*, used for notifying the listener that a service has been discovered, and *SearchFinished*, which notifies that the search has been finished, either explicitly by an application or due to some error.

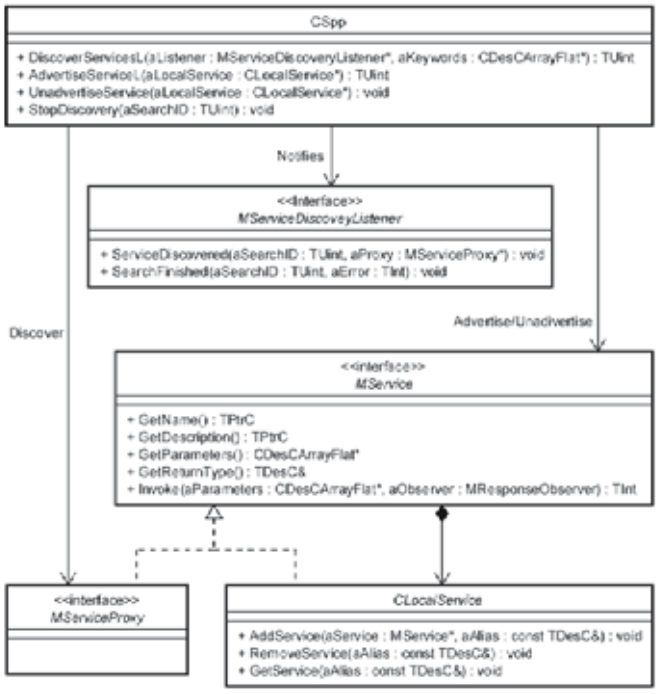


Fig. 2. UML diagram of the SPPs implementation.

C. Host Discovery Plug-in

The Host Discovery Plug-in, which is also part of the Pervasive Networking Module, is used to search hosts in the pervasive environment so that the communication can be established among them. The most important classes of this plug-in are illustrated in the class diagram of Figure 3 and detailed next.

- *CHdp*: this abstract interface represents a Host Discovery Plug-in with one factory method, *NewL*, and two abstract methods, *DiscoverHosts* and *StopDiscovery*, which implements its functionalities. The *DiscoverHosts* method enables devices to search for hosts in the pervasive environment through the protocol associated with the plug-in. This method also returns a search identifier, enabling applications to stop the search whenever they need. The remaining method of the HDP, *StopDiscovery*, is used to cancel a search for hosts, and receives a search identifier as a parameter.
- *MHostDiscoveryListener*: this interface is used to receive notifications about the host discovery process. It specifies

two methods, one for receiving a recently discovered host, *HostDiscovered*, and another for indicating the search has ended, *SearchFinished*. The *HostDiscovered* method receives in the parameters the host that has been discovered, specified by the *MRemoteHost* interface, and the identifier of the search associated with the discovery. The *SearchFinished* method, on the other hand, receives a code which identifies if the search has been finished due to an application request, that is normally, or due to an error.

- *MRemoteHost*: this interface represents a remote host, providing methods for obtaining its name and description, and for opening a connection with it. Also, it is possible to retrieve the services provided by this remote host through the *GetProvidedServices* method. The *ServiceDiscovered* method of this instance is used by the remote host to pass the services it provides, each one as an instance of *MServiceProxy*.

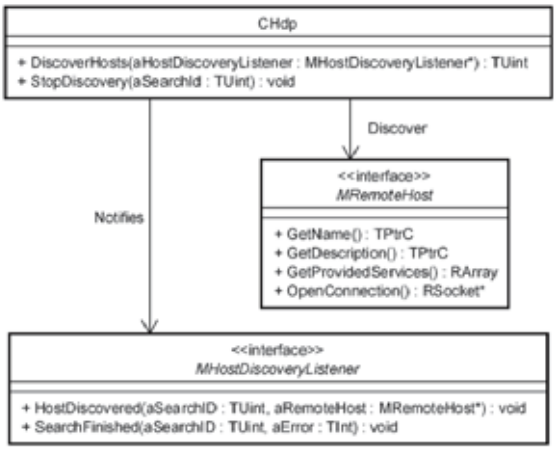


Fig. 3. UML diagram of the HDP implementation.

D. Context Awareness Plug-in

Context Awareness Plug-ins, CAPs, are used to retrieve context information from the pervasive environment. This information can be of any type, for example, the temperature of a room, the location of a user in the environment, and so on. This section presents the Context Awareness Plug-ins, describing its main classes and interfaces, which are illustrated in the class diagram of Figure 4.

- *CCap*: this is the abstract interface that represents a Context Awareness Plug-in, providing one factory method, and three functionalities specified as abstract methods: *RetrieveContextInformation*, *RegisterContext Listener*, and *UnregisterContextListener*. The first one enables its users to retrieve context information using the key-value approach presented in Section II. The *RegisterContextListener* is used for registering a listener that will receive notifications about certain context events, which are fired by the CAP. It receives a condition, as an instance of the *MContextCondition* interface, which determines when the event should be fired, and an instance of the *MContextEventListener* interface, which receives the event once the condition is satisfied. Also, this method returns a registration identifier so that it can be revoked through the *UnregisterContextListener* method.
- *MContextCondition*: this interface represents the condition that regulates the triggering of a context event. This condition is passed when registering an context listener to a condition. The satisfiability of the condition is checked through the *IsSatisfied* method.
- *CContextEvent*: this class represents a context event fired by a CAP. A context event provides two information, the key and current value of the context information they are associated, obtained respectively through the *GetContextInformationKey* and *GetContextInformationValue* methods.
- *MContextEventListener*: this interface represents the listener of the context events. When an event is fired, the listeners are notified through the *ReceiveContextEvent* method, which receives a *CContextEvent* object representing the context event fired by the CAP.
- *CConditionMonitor*: this class is responsible for monitoring a context condition registered within a CAP. It is implemented as an Active Object and creates a *CContextEvent* object every time the context condition is satisfied. This object will then be passed to the listener associated with the condition (i.e., instance of *MContextEventListener*).

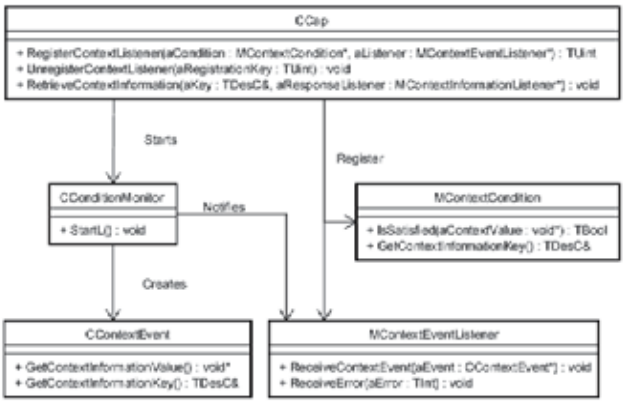


Fig. 4. UML diagram of the CAP implementation.

E. Middleware Facade

As we mentioned in Section II, the Middleware Facade Module is the point through which applications can use the middleware functionalities. Thus, methods specified by each plug-in interface are also present in the Middleware Facade. When an application invokes a method of the facade associated

with the plug-ins defined in the Pervasive Networking Module, it forwards the request to the respective method of all related plug-ins currently loaded. A search for services, for example, has to be propagated throughout all the available SPPs. On the other hand, when invoking a method associated with the plug-ins defined in the Context Awareness Module, the facade will search for the plug-in that provides the context information or context event required. Thus, the request is passed only to this plug-in. The Middleware Facade also manages the plug-ins through the use of the ECOM Framework. More precisely, when a new plug-in is installed, the facade is notified by the ECOM server, so that the plug-in can be loaded and then used by the applications.

IV. Application Scenario

In this section we present an application scenario for illustrating the applicability of *Wings4Symbian* middleware. The basic idea is to define how the *Wings4Symbian* middleware can be dynamically reconfigured taking into account the plug-ins installed, and also how developers can use the *Wings4Symbian* API to develop pervasive applications.

A. Book Searcher Application

This application was implemented to provide access to the Embedded Systems and Pervasive Computing Laboratory Library, to enable users to search and/or be notified of specific books that are available, according to they preferences. For this application the service discovery capability provided by *Wings4Symbian* middleware is used. This functionality works in the following way. The user passes a set of keywords to the application provided as an instance variable member, *iBookKeywords* of *CBookSearcher*. The application tries to discover the service that performs the search for books through a method *SearchBooks*. This method uses an array of service keywords specified as a member variable (*iServiceKeywords*) containing the words *books* and *search* that specify the service to be searched. The application retrieves the discovered services (method *ServiceDiscovered*), and from then obtains the one which performs the search for books. If such a service has been found the search is stopped and the application invoke it, passing the keywords provided by the user (*iBookKeywords*). On the server side, these keywords are checked against each book title of the library. Those that are relevant are returned to the application, which shows them to the user.

Besides searching for books, the user can register his interest on a specific book or a subject. As soon as books that satisfy his preferences are available and he is in laboratory building, he is notified about the availability of the books.

B. Reconfiguring the Middleware

Middleware reconfiguration is performed through the addition and removal of plug-ins. Each of these processes can be split in two steps: installation and loading, for the addition, and uninstallation and unloading, for the removal. The installation consists of downloading the plug-in file to a specific folder of the Symbian OS. The ECOM framework, thus, detect that a plug-in has been inserted into such a folder, and then notify the Middleware Facade, as we explained in Section III-E. After that, the facade then loads the recently installed plug-in. This loading process is achieved by sending requests to the ECOM server. The ECOM server instantiates the plug-in implementation and loads it in the process of the calling application. Different applications running simultaneously in different processes can load the same plug-in, and thus, the ECOM server makes use of reference counting in order to unload the plug-in automatically when all applications have released it. Therefore, when no application is making use of a plug-in, it can be uninstalled, by removing its file from the Symbian OS plug-in folder.

Speaking specifically about our application scenario, when the user enters a new environment for the first time, the applications available in it will be downloaded to his/her device along with the plug-ins it depends on. More precisely, five steps are executed: 1) download the applications available, 2) download the plug-ins required by each application, considering the device settings where they will run, 3) install the applications, 4) install the plug-ins, according to the process described above, and finally 5) execute the applications.

V. Related Works

On the field of middlewares for pervasive computing, it is a fact that a reasonable number of works have been proposed. One of these works is MARKS [11] (Middleware Adaptability for Resource Discovery, Knowledge Usability and Self-healing), a middleware which aims to incorporate less-explored areas of pervasive computing, specifically knowledge usability. However, the first prototypal implementation offers just Device Discovery, Service Discovery, Context-Awareness, and Self-healing services. This prototype was implemented using the Visual Studio .NET compact framework and does not offer a simple and flexible mechanism to dynamically load new features to the middleware in a plug-in based way, as in the *Wings4Symbian* middleware.

O3MiSCID [12] (Object-Oriented Opensource Middleware for Services Connection, Inspection and Discovery) is a pervasive middleware built in three layers, which are responsible for network communication, handle of services and doing semantic description of services (ontology) at high layer. O3MiSCID handles services (declare and discover) using the DNS-SD (DNS-Service Discovery) mechanism and it has been implemented using C++, Java and Tcl. Therefore, O3MiSCID is a middleware concerned just with connection, inspection and discovery of services, not worried about context-awareness or dynamically addition of new features.

Cortex [13] is a middleware for context awareness in pervasive and ad hoc environments comprised of a set of component frameworks, each of them targeted at a specific purpose, defining the kind of component it accepts. The component frameworks defined by the *Cortex* middleware are: *service*

discovery framework, *context framework*, *publish-subscribe framework*, and *resource management framework*. In a broad sense, these frameworks are respectively responsible for the dissemination of events, discovery of services, acquisition of context information, and management of local resources like CPU, memory, and network connections. These component frameworks are implemented over the *OpenCOM* component model, so that components can be inserted and removed from the middleware on the fly. The *Cortex* middleware, however, does not provide host discovery features.

SOCAM [14] is an ontology and service-based middleware for context awareness in pervasive environments. The architecture of the *SOCAM* middleware is based on a set of components responsible for acquiring and representing context information (*Context providers*), interpreting them (*Context interpreters*), and finally passing them to the so-called *Context awareness services*. Context information is provided through services, so that context information can be acquired by remote hosts in a decoupled way. The focus of the SOCAM middleware is too much on context awareness, not dealing specifically with dynamic reconfiguration aspects.

VI. Final Remarks

In this paper we presented a pervasive computing middleware implementation for smartphones running the Symbian Operating System, named *Wings4Symbian*. The infrastructure is focused on service provision, host discovery, and context awareness. Also, the dynamic extension of the middleware was also tackled, through the use of a plug-in-based architecture and the ECOM framework. Therefore, the middleware provides the necessary functionalities to build applications for pervasive environments, with the possibility of being dynamically reconfigured. We also presented an application scenario, which showed not only how the middleware can be adapted according to the environment, but also how its API can be used to develop pervasive computing applications.

As a future work, our goal is to build a set of plug-ins so that developers can use them in their own applications scenarios. So far, we developed four plug-ins that are currently being used, discussed in Section IV two *Service Provision Plug-ins* (SPPs), one *Host Discovery Plug-in* (HDP), and one *Context Awareness Plug-in* (CAP). The SPPs are implemented using the Bluetooth Service Discovery Protocol (SDP) and the Universal Plug and Play (UPnP) protocol. For the latter, the CyberLink [1] API is being used. The HDP is also based on the Bluetooth. Finally, the CAP is targeted at providing information related to the Embedded Systems and Pervasive Computing Laboratory Library, such as people present in the building, books that are available in the library, among others.

Acknowledgments

The authors would like to thank Nokia do Brasil and Instituto Nokia de Tecnologia for the partial support to develop this work.

[1] http://www.cybergarage.org

References

[1] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 94–104, September 1991.

[2] D. Saha and A. Mukherjee, "Pervasive computing: A paradigm for the 21st century," *Computer*, vol. 36, no. 3, pp. 25–31, March 2003.

[3] G. Roussos, L. Iftode, and H. Mitchell, "Guest editors' introduction: The smart phone–a first platform for pervasive computing," *IEEE Pervasive Computing*, vol. 4, no. 2, pp. 18–19, 2005.

[4] G. D. Abowd, A. J. Marsh, and S. Maglavera, "Enabling pervasive computing with smart phones," *IEEE Pervasive Computing*, vol. 4, no. 2, pp. 20–25, 2005.

[5] N. Ravi, P. Stern, N. Desai, and L. Iftode, "Accessing ubiquitous services using smart phones." in *PerCom*, 2005, pp. 383–393.

[6] J. Sales, A. Rogers, A. Thoelke, C. Freitas, C. Dive-Reclus, D. May, D. Feather, H. Morgan, P. Scobie, J. Strong, J. Parker, S. Williams, and T. Lofthouse, *Symbian OS Internals*. England: John Wiley and Sons, 2005.

[7] J. Stichbury, *Symbian OS Explained*. England: Wiley, 2005.

[8] L. Edwards, *Developing Series 60 Applications*. Addison-Wesley, 2004.

[9] D. Saha, A. Mukherjee, and S. Bandyopadhyay, *Networking Infrastructure for Pervasive Computing*. Boston, USA: Kluwer Academic Publishers, 2003.

[10] J. Mayer, I. Melzer, and F. Schweiggert, "Lightweight plug-in-based application development," in *Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World*, M. Aksit, M. Mezini, and R. Unland, Eds. Erfurt, Alemanha: Springer-Verlag, 2003, pp. 87–102.

[11] M. Sharmin, S. Ahmed, and S. I. Ahamed, "MARKS (Middleware Adaptability for Resource Discovery, Knowledge Usability and Self-healing) for mobile devices of pervasive computing environments," in *Proceedings of the Third International Conference on Information Technology : New Generations*, Las Vegas, Nevada, USA, April 2006, pp. 306–313.

[12] R. Emonet, D. Vaufreydaz, P. Reignier, and J. Letessier, "O3miscid: an object oriented opensource middleware for service connection, introspection and discovery," in *1st IEEE International Workshop on Services Integration in Pervasive Environments*, June 2006.

[13] C.-F. Srensen, M. Wu, T. Sivaharan, G. S. Blair, P. Okanda, A. Friday, and H. Duran-Limon, "A context-aware middleware for applications in mobile ad hoc environments," in *Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing*. New York, NY, USA: ACM Press, 2004, pp. 107–110.

[14] T. Gu, H. K. Pung, and D. Q. Zhang, "A service-oriented middleware for building context-aware services," *Journal of Network and Computer Applications*, vol. 28, no. 1, pp. 1–18, 2005.

An OWL/SWRL based Diagnosis Approach in a Pervasive Middleware

Weishan Zhang and Klaus Marius Hansen
Department of Computer Science, University of Aarhus
Aabogade 34, 8200 Århus N, Denmark
{zhangws,klaus.m.hansen}@daimi.au.dk

Abstract

Diagnosis is the most important step for achieving self-healing of systems, which is a challenge in pervasive computing. In this paper, we present a semantic, state machine-based diagnosis approach for a web-service based middleware. We use OWL ontologies and SWRL to develop both diagnosis and monitoring rules, based on state changes and also invocation relationships. Malfunction information and its resolution are encoded in an OWL ontology as a part of a Device ontology, and can be used at run time to check how to resolve malfunction, and further to fulfill self-healing activities. SWRL rules at both device level and system level are designed and will be executed as needed. The evaluations in terms of extensibility, performance and scalability show that this approach is effective in pervasive service environment.

1 Introduction and Motivation

Web services are increasingly needed to be adopted as service provision mechanisms in pervasive computing environment. This trend is exemplified during the inauguration phase of the *Hydra* project(IST-2005-034891), by some companies that donate us Zigbee devices and other embedded devices that enabling pervasive computing, and express their wishes for web service enabled devices.

A concrete agriculture scenario that we are considering in the Hydra project is as followed:
Bjarne is an agricultural worker at a large pig farm in Denmark. As he walks through the pens to check whether the pigs are provided with correct amount of food, his work is interrupted by a sound from his PDA, indicating that a high priority alarm has arrived. Apparently, the ventilation system in the pig stable has malfunctioned. After acknowledging the alarm and the system begins to diagnosis and soon it decides that the cause of the problem is 'power supply off because of fuse blown'. Then he can prepare a fuse and repair the ventilator. After repairing it, he signs off the alarm, and writes a log on what he has done.

As can be seen from the above scenario, it is very important that the Hydra middleware can provide diagnosis functionality to the end user, or better to achieve self-healing when there is malfunction. Such kind of self-healing can not be always finished automatically, for example device down because of fuse broken. But providing diagnosis and then resolution suggestions would be the most important step towards malfunction recovery.

In this paper, we present an OWL ontology (the Web Ontology language)[1] and SWRL (Semantic Web Rule Language)[2] based diagnosis using state machine and sniffering of process invocation in the context of the Hydra middleware. The malfunction information and its resolution are encoded in an OWL ontology as part of a Device ontology, and can be used at run time to check appropriate resolution to the malfunction, and further to fulfill self-healing activities. We use SWRL to develop monitoring and diagnosis rules, and these rules, together with OWL ontologies, can help make intelligent decisions on where malfunction occurs and its resolution.

The rest of the paper is structured as follows: Section 2 presents an overview of the Hydra middleware; We then show the diagnosis ontologies used in Hydra; In section 4, design of both rules and the Diagnosis Manager are presented. Section 5 evaluates our work with the extensibility, performance and scalability. We compare our work with the related work in section 6. Conclusions and future work end the paper.

2 Web service based middleware-Hydra

The Hydra project is developing a service-oriented and self-managed middleware for pervasive embedded and network systems based on web service. According to the available resources, the function structure of the Hydra middleware is divided into two parts, namely Application El-

[1] OWL Web Ontology Language Guide. http://www.w3.org/TR/owl-guide/

[2] SWRL specification homepage. http://www.w3.org/Submission/SWRL/

ements(AEs) and Device Elements(DEs). AEs are meant to be running on powerful machines, DEs describe components that are usually deployed inside Hydra-enabled devices where small devices maybe involved. The Layered architecture of the Hydra middleware is shown in Figure 1.

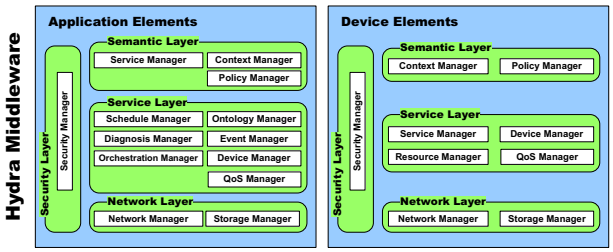


Figure 1. Hydra middleware Layered architecture

Diagnosis Manager is used to monitor the system conditions and states in order to fulfill error detection and logging device events. Its functions include system diagnosis and device diagnosis.

The Event Manager is used to provide publish/subscribe functionality to the HYDRA middleware. In general, publish/subscribe communication as provided by the Event Manager provides an application-level, selective multicast that decouples senders and receivers in time, space and data.

3 Ontologies used in the Diagnosis Manager

There are several ontologies involved in the diagnosis process, namely Device ontology, Malfunction ontology, and StateMachine ontology. The DeviceRule ontology is used for holding all diagnosis rules as introduced in Section 4.1. The high level structure of the diagnosis ontologies is shown in Figure 2.

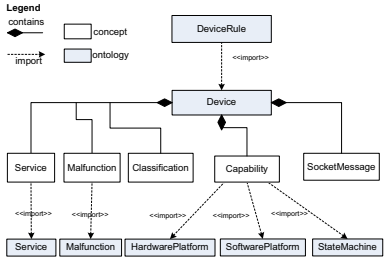


Figure 2. Diagnosis ontologies structure

The Device ontology is used to define some basic information of a Hydra device, for example device type classification(e.g. mobile phone, sensor), device model and manufacturer, and so on. The device type classification is based mainly on AMIGO project ontologies [7]. To facilitate diagnosis, there is a concept called *HydraSystem* to model a system composed of devices to provide services. And there is a corresponding object property *hasDevice* which has the domain of *HydraSystem* and range as *HydraDevice*. There are also concepts used for the monitoring of web service calls, including *SocketProcess*, *SocketMessage* and *IPAddress*. The *HydraDevice* concept has a data type property *currentMalfunction* which is used to store the inferred device malfunction diagnosis information at run time and will be exemplified later.

To enable state based diagnosis, a state machine ontology is developed based on [5] with many improvements: firstly, we add a datatype property *isCurrent* in order to indicate whether a state is current or not; secondly, we add a *doActivity* object property to the *State* in order to specify the corresponding activity in a state and this makes the state machine complete; thirdly, we add a datatype property *hasResult* to the *Action* (including activity) concept in order to check the execution result at run time. Three other datatype properties are also added to model historial action results. This facilitates the specification of diagnosis rule based on state and activity result and its history.

The device Malfunction ontology is used to model malfunction and recovery resolutions. We separate the malfunctions into two categories: *Error* (including device totally down) and *Warning* (including function scale-down, and plain warning). There are also two other concepts, *Cause* and *Remedy*, which are used to describe the origin of malfunction and its resolution.

A more detailed but simplified view of the ontologies used in the diagnosis is depicted in Figure 3.

4 Design of the Diagnosis Manager

Hydra implements a service-oriented architecture based on web service interaction among devices. Thus a reasonable granularity to build a self-management system on is the level of web service requests and responses. Furthermore, we are interested in the states of devices per se, i.e., is the device operational, stopped, not working and if it is operational what is the value of its sensor readings (if any) or its actuator state (if any). This leads us initially to focus on status reporting of the following two forms:

- *State change reporting.* State machines are used to report their state changes as events through the Hydra Event Manager.
- *Web service request/reply reporting.* The requests and replies (and their associated data) can be used to analyse the runtime structure of the Hydra systems. Here a tool called IPSniffer is used to report invocations.

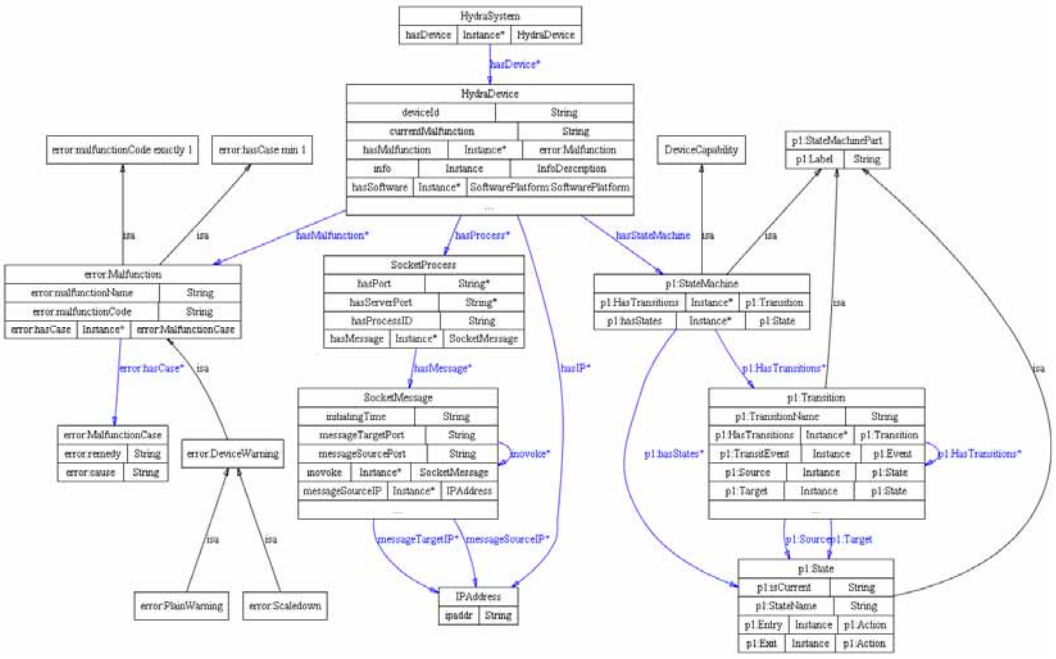


Figure 3. Partial details of the Diagnosis Manager used ontologies

4.1 Design of SWRL rules

Diagnosis is a complex task which need intelligence to infer what is the reason for error and its consequence. The OWL-DL ontologies themselves are hardly expressive enough to specify diagnosis rules. As an alliance to OWL, SWRL can be used to write rules to reason about OWL individuals and to infer new knowledge about those individuals. A SWRL rule means that if all the atoms in the antecedent (body) are true, then the consequent (head) must also be true. In the SWRL rules, the symbol $\wedge$ means conjunction, $?x$ stands for a variable, $\rightarrow$ means implication; and if there is no ? in the variable, then it is an instance.

Device level rules

Device level rules are used for a certain type of devices which are supposed to be generic for that type of devices. The followed is an example of mobile phone battery monitoring, if battery level is less than 10%, a warning will be published.

$device : MobilePhone(?device) \wedge$
$device : hasHardware(?device, ?hardware) \wedge$
$Hardware : primaryBattery(?hardware, ?battery) \wedge$
$Hardware : batteryLevel(?battery, ?level) \wedge$
$swrlb : lessThanOrEqual(?level, 0.1)$
$\rightarrow VeryLowBattery(?device)$

Another monitoring rule is if the flow measured from the flowmeter is more than 16 (gallon/minute), then it is too high and should be repaired as soon as possible:

$device : FlowMeter(?device) \wedge$
$device : hasStateMachine(?device, ?sm) \wedge$
$statemachine : hasStates(?sm, ?state) \wedge$
$statemachine : doActivity(?state, ?action) \wedge$
$statemachine : actionResult(?action, ?r) \wedge$
$abox : isNumeric(?r) \wedge swrlb : greaterThan(?r, 16.0) \rightarrow$
$device : currentMalfunction(device : Flowmeter, "FlowHigh")$

The rule for IPSniffer is used for both checking process id, ip address, port etc. and inferring invoking relationships.

$device : messageSourceIP(?message1, ?ip1) \wedge$
$device : ipaddr(?ip1, ?ipa1) \wedge$
$device : messageSourcePort(?message1, ?port1) \wedge$
$device : hasMessage(?process1, ?message1) \wedge$
$device : hasProcessID(?process1, ?pid1) \wedge$
$device : messageTargetIP(?message2, ?ip2) \wedge$
$device : messageSourceIP(?message2, ?ip3) \wedge$
$device : ipaddr(?ip3, ?ipa3) \wedge$
$device : messageTargetPort(?message2, ?port2) \wedge$
$device : hasMessage(?process2, ?message2) \wedge$
$device : hasProcessID(?process2, ?pid2) \wedge$
$swrlb : equal(?port1, ?port2) \wedge$
$device : initiatingTime(?message1, ?time1) \wedge$
$device : initiatingTime(?message2, ?t2) \wedge$
$temporal : duration(?d, ?time1, ?t2, temporal : Milliseconds)$
$\wedge swrlb : lessThanOrEqual(?d, 60000)$
$\rightarrow device : inovoke(?message1, ?message2) \wedge$

$sqwrl : select(?ipa1, ?port1, ?pid1, ?ipa3, ?port2, ?pid2, ?time1)$

System level rules

System level rules are used to specify rules span multiple devices in a system. In the introduced agriculture scenario, thermometers are used to measure both indoor and outdoor temperature, which are named PicoTh03_Outdoor and PicoTh03_Indoor respectively. In the summer time, when outdoor temperature is between 12 and 33 degree, the indoor should follow the same trend as the outdoor temperature. Or else, we can infer that the ventilator is down.

$device : hasStateMachine(device : PicoTh03_Outdoor, ?sm)$
$\wedge\ statemachine : hasStates(?sm, ?state) \wedge$
$statemachine : doActivity(?state, ?action) \wedge$
$statemachine : actionResult(?action, ?r) \wedge$
$statemachine : historicalResult1(?action, ?r1) \wedge$
$statemachine : historicalResult2(?action, ?r2) \wedge$
$statemachine : historicalResult3(?action, ?r3) \wedge$
$swrlb : add(?tempaverage, ?r1, ?r2, ?r3) \wedge$
$swrlb : divide(?average, ?tempaverage, 3) \wedge$
$swrlb : subtract(?temp1, ?r, ?r1) \wedge$
$swrlb : subtract(?temp2, ?r1, ?r2) \wedge$
$swrlb : subtract(?temp3, ?r2, ?r3) \wedge$
$swrlb : add(?temp, ?temp1, ?temp2, ?temp3) \wedge$
$swrlb : greaterThan(?average, 12.0) \wedge$
$swrlb : lessThan(?average, 33.0) \wedge$
$swrlb : lessThan(?temp, 0) \wedge$
$device : hasStateMachine(device : PicoTh03_Indoor, ?sm_b)$
$\wedge\ statemachine : hasStates(?sm_b, ?state_b) \wedge$
$statemachine : doActivity(?state_b, ?action_b) \wedge$
$statemachine : actionResult(?action_b, ?r_b) \wedge$
$statemachine : historicalResult1(?action_b, ?r1_b) \wedge$
$statemachine : historicalResult2(?action_b, ?r2_b) \wedge$
$statemachine : historicalResult3(?action_b, ?r3_b) \wedge$
$swrlb : subtract(?temp1_b, ?r_b, ?r1_b) \wedge$
$swrlb : subtract(?temp2_b, ?r1_b, ?r2_b) \wedge$
$swrlb : subtract(?temp3_b, ?r2_b, ?r3_b) \wedge$
$swrlb : add(?temp_b, ?temp1_b, ?temp2_b, ?temp3_b) \wedge$
$swrlb : greaterThan(?temp_b, 0) \rightarrow device : currentMalfunction(device : VentilatorMY0193, "VentilatorDown")$

The processing of this rule will get the trend with the difference of continuous temperature measuring of indoor and outdoor temperature, and also an instance of the property ("VentilatorDown") *currentMalfunction* of concept *HydraDevice* (which is VentilatorMY0193) will be inferred. Then the Malfunction ontology will be checked for the resolution of the problem based on the malfunction cause. In our case, Malfunction ontology gives us the solution as the "power supply off because of fuse blown".

Usage of Malfunction and Device ontology

For example, Bjarne get a warning of "Grundfos-PumpMQ345 failed to start", which is a high priority task for him as the pump is used for feeding the pigs. A diagnosis task is initiated to check what is wrong with the pump, but as a newly installed pump, there is still no error resolution to this model of pump in the Malfunction ontology. As a further step, the diagnosis system will conduct subsumption reasoning and search for the device *Type* in the Device ontology, which is found as *FluidPump*, and then its manufacturer is also queried. Now another query to the Device ontology will get a similar pump called *GrundfosPumpMQ335* as of the same type from the same manufacturer "Grundfos". And based on the name of the error and pump type, the solution from a query to Malfunction ontology is suggested "replace a capacitor", which is happily the solution to solve the problem.

4.2 Diagnosis manager architecture

Based on the current diagnosis requirements, and also the status of OWL/SWRL, we come up with the following architecture for the Diagnosis Manager as shown in Figure 4, in which the *Component Control* and *Change Management* are enclosed with dashed line, taken Kramer and Magee [6] three Layered architecture as a reference model.

The bottom of the architecture is the ontologies/rules, in which knowledge of devices, and state based diagnosis are encoded. When there are state change events, the device state machine instance in the state machine ontology need to be updated, and also these state changes will be published with state machine state changes as event topic. The Diagnosis Manager is an event subscriber to the state machine state change events, it will then update the corresponding state instances in the ontology. At the same time, this will trigger the diagnosis of the device status, executing the SWRL rules to monitor the health status of devices, and also trigger the reasoning of possible device errors and their resolutions. The Diagnosis Manager will publish the diagnosis results as an event publisher.

The Diagnosis Manager mainly runs on powerful PC or a proxy for an embedded device running on a powerful node. For those node with limited capabilities, only state will be reported, which can delegate its own diagnosis to other node or its proxy.

For the actual implementation, we adopted a mix of the Blackboard architecture style and the Layered architecture, and use the observer pattern in both the updating of state machine ontology and inferred result parsing.

Figure 4. Diagnosis Manager architecture

5 Evaluation

5.1 Extensibility

At present, the extensibility is evaluated by the applicability to new devices added to a system. We started the development of Diagnosis Manager with the rule for temperature monitoring. After finishing the implementation and testing, we then try to handle the flowmeter diagnosis rules. The steps involved are:

1. Add the flowmeter device to the *HydraSystem* concept instance called "Pig" in the Device ontology.

2. Add the flowmeter state machine instance to the StateMachine ontology.

3. Add the flowmeter state machine instance to the hasStateMachine property of the "flowmeter" device.

4. Add flowmeter diagnosis rule to the DeviceRule ontology.

After this, we test the Diagnosis Manager and it runs very well. No single line of Diagnosis Manager code needs to be changed. In summary, the adding of new devices to a certain system is very straightforward. The adding of new devices can be at run time, if the rules for the new devices are existing, then the diagnosis process can be directly working for the new devices.

5.2 Performance

The following software platform is used for measuring performance: Protege 3.4 Build 125, JVM 1.6.02-b06, Heap memory is 266M, Windows Vista. The hardware platform is: Thinkpad T60 Core2Duo 2G CPU, 7200rpm harddisk, 2G DDR2 RAM. The time measurement is in millisecond. The size of DeviceRule ontology is 210,394 bytes, and contains 22 rules.

We measured the performance as shown in Table 1. An interesting thing is after some time of running, the Diagnosis Manager is running stably with the total time in 260-270 ms for processing an event, a bit faster than the one when it starts. Here the parsing of the inferred result is running in a multi-threaded way in the Diagnosis Manager.

Update	InferringTime	AfterEventTillInferred
383	380	382
322	319	321
282	278	282
272	269	271
265	263	265
270	267	269
268	266	269

Table 1. Diagnosis Manager performance

5.3 Scalability

The scalability is evaluated through clients continuously publishing their states (thermometers and flowmeters) as events, in an almost parallel way and each of the client sends messages as fast as possible in a loop. Then we measure how long it will be, starting from the publishing till the end of inferring and publish related inferring result. Time needed (y-axis) is shown in Figure 5 (x-axis shows the number of events) . We can see that the time taken is in linear with the events need to be processed.

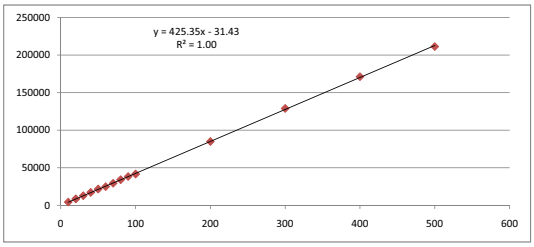


Figure 5. Diagnosis Manager scalability

6 Related work

Kramer and Magee [6] recently proposed a reference model for self-managed systems, which is composed of component control, change management and goal management. In this paper, we largely followed this work for the Layered architecture, but mainly focus the component control and change management. At the same time, a mix of Blackboard architecture and Layered architecture are applied to improve performance and extensibility.

Self-healing is one of the main challenges to autonomic pervasive computing. Generally speaking, our approach applied the same idea of ETS [2], in terms of the using of states for detecting source of failure, and then notification

of failure source. And this process is actually universal for error detections. Our ontology and SWRL rule based approach provides a way of intelligent detection and resolution, which is not easily achievable by ETS.

Work in [1] shares some similarity with us on the usage of semantic web approach for achieving self-managing. Our approach is non-intrusive, SWRL rules are automatically executed using state machine instead of explicitly inserting sensor code to program, and is more suitable for the characteristics of pervasive devices.

Various failures in a pervasive system are classified in [4], and an architecture for fault tolerant pervasive computing is proposed. We focus not only on device failure monitoring using the device state machine, but also system level detection using the relationships of different state machine instances. In addition, our approach can be more intelligent in terms that ontology reasoning can help the diagnosis.

There are many researches from traditional artificial intelligence point of view dealing with the diagnosis in various field, e.g. [3]. These traditional approaches are not utilizing the context ontologies that are already there in pervasive systems and are used for context-awareness and other purposes. The open world assumption in OWL/SWRL and hence in our approach makes our proposed approach well suited for the openness of the pervasive computing environment, which automatically rejects the approaches using Prolog kind of rules which use close world assumption.

7 Conclusions and future work

OWL/SWRL is adopting an open world assumption which is in nature very suitable for the pervasive computing systems, where the openness and dynamicity dominate the interaction and function. OWL is widely used in pervasive computing, for the purpose of context awareness, service selection and composition. The potentials of OWL and context awareness could be further extended as we have shown in this paper.

Diagnosis is the most important step for achieving self-healing, which is a challenge in pervasive computing. We present a semantic and state machine based diagnosis approach using OWL ontology and SWRL, for the Hydra middleware. The malfunction information and its resolution are encoded in an OWL ontology, and can be used at run time to infer the solution to the malfunction, and further to fulfill self-healing activities. SWRL is used to develop monitoring and diagnosis rules, which can help make intelligent decisions when there is malfunction occurs. IPSniffer will help diagnosis on devices that are dead or no response which provides fault tolerance in our approach.

The evaluations relieved us for the worrying of performance of the OWL/SWRL based Diagnosis Manager. In order to improve performance, we followed a mix of both the Blackboard architecture style and the Layered architecture style. The evaluations show that the Diagnosis Manager is usable in terms of extensibility, performance and scalability. The proposed approach provides an uniform, coherent and natural way to fully utilize the existing OWL/SWRL reasoning power, and extend it for considering the dynamic aspects of the pervasive system for diagnosis, which is very suitable for the characteristics of the pervasive computing environment.

We are improving the IPSniffer based diagnosis that only reports invocation relationships at present. The integration with security manager and ontology manager are under way. Probability in OWL/SWRL is to be added in the future to make the diagnosis more intelligent. More experiments in a larger scale will be conducted for testing the resolving of rule conflicts, accuracy of diagnosis and so on.

Acknowledgements

The research reported in this paper has been supported by the Hydra EU project (IST-2005-034891).

References

[1] B. J. O. A. R. Haydarlou, M. A. Oey and F. M. T. Brazier. Use-case driven approach to self-monitoring in autonomic systems. *The Third International Conference on Autonomic and Autonomous Systems*, 2007.

[2] S. Ahmed, M. Sharmin, and S. Ahamed. ETS (Efficient, Transparent, and Secured) Self-healing Service for Pervasive Computing Applications. *International Journal of Network Security*, 4(3):271–281, 2007.

[3] R. Barco, L. Díez, V. Wille, and P. Lázaro. Automatic diagnosis of mobile communication networks under imprecise parameters. *Expert Systems With Applications*, 2007.

[4] S. Chetan, A. Ranganathan, and R. Campbell. Towards fault tolerant pervasive computing. *Technology and Society Magazine, IEEE*, 24(1):38–44, 2005.

[5] P. Dolog. Model-driven navigation design for semantic web applications with the uml-guide. *Engineering Advanced Web Applications, In Maristella Matera and Sara Comai (eds.)*, Dec. 2004.

[6] J. Kramer and J. Magee. Self-Managed Systems: an Architectural Challenge. *International Conference on Software Engineering*, pages 259–268, 2007.

[7] I. A. Project. Amigo middleware core: Prototype implementation and documentation, deliverable 3.2. In *IST-2004-004182*, 2006.

A Constraint Model for Automated Deployment of Automotive Control Software

Mihai Nica Bernhard Peischl Franz Wotawa*

Institute for Software Technology
Graz University of Technology
8010 Graz, Austria
{mnica,bpeischl,wotawa}@ist.tugraz.at

Abstract

In this paper we address automated software deployment for embedded automotive systems in terms of a constraint satisfaction problem (CSP). Our purely model-based approach allows for fully automatic deployment of software functions in a resource-constrained system (exemplified in terms of memory and bus load). Besides of its applicability in an early stage of development, most notably, our model incorporates optimization criteria from algorithmic approaches proposed recently. Capturing the problem-relevant aspects in terms of a CSP is straightforward and thus easily extendable to complex scenarios like, for example, temporal requirements or the diverse bus protocols in the automotive domain.

1 Introduction

Today's upper class cars contain up to 80 ECUs (Electronic Control Units), several bus systems, and about 55 percent of all failures are caused by electronics, software, cables and connectors [1], [2]. More and more functions in today's cars involve electronics and software, 80-90 percent of the new innovative features are realized by distributed embedded systems. Following this mainstream trend, even highly safety critical mechanical and hydraulic control systems will be replaced by electronic components.

In recent years, the focus in engineering embedded automotive systems has been on rather detailed abstractions primarily dealing with implementation related issues like models for code generation. Model-based optimization techniques typically take a back seat in the overall design process since they lack suitable, standardized notations, methodologies, and integration into the model-driven tool chain.

As today's embedded automotive software is highly distributed, the automotive industry devotes increasing efforts to develop tools for automated software deployment [3]. The underlying foundations comprise techniques like genetic algorithms and various other clustering techniques [3]. However, to our best knowledge, none of the current approaches addresses automated software deployment in terms of a model-based approach. Relying on an algorithmic approach one has to perform measurements to obtain meaningful metrics for certain parameters as, for example, a reference value for the bus load. Besides of the (often) painstaking provision of a prototype to obtain concrete measurements, this considerably hampers the seamless integration into the model-based development paradigm.

In this paper we address the prevalent complexity of automated software deployment in a resource-constrained setting even catering stakeholders at an early development stage, where no reference measurements for a concrete ECU might be available. Our approach relies on modeling software deployment in terms of a constraint satisfaction problem (CSP). Most notably, this model allows for incorporating optimization criteria from algorithmic approaches proposed recently [4]. Moreover, the model-based approach to automated software deployment directly supports an iterative refinement of the model down to the level of protocols, gateways, of software drivers.

This article is organized as follows. In Chapter 2 we present the CSP and its associated parameters, in Chapter 3 we explain how the partitioning problem is solved by means of a CSP, in Chapter 4 we discuss some constraint solvers available on the market. Finally Chapter 5 presents our conclusions and discusses related work.

2 Constraint Satisfaction Problem

Constraint systems are a natural and straightforward way of describing specifications and requirements for hardware and software systems. A *Constraint Satisfaction Problem*

*Authors are listed in alphabetical order. The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsfrderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT) and by the "Fonds zur Förderung der wissenschaftlichen Forschung"(FWF). Special thanks to the reviewers and to our colleague Willibald Krenn for their useful feedback.

```
1.  { var_1 = ( x_0 < y_0 );
2.  min_1 = x_0;
3.  min_2 = y_0; }
```

Figure 1. Program for computing the minimum between two numbers

$$\begin{aligned} \textbf{Variables:} \quad & V = \{var\_1, x\_0, y\_0, min\_1, min\_2\} \\ \textbf{Domains:} \quad & D = \{D(x) = \mathbb{N} | x \in V\} \\ \textbf{Constraints:} \quad & \\ & CO = \left\{ \begin{array}{c} var\_1 = (x\_0 < y\_0), \\ min\_1 = x\_0, min\_2 = y\_0 \end{array} \right\} \end{aligned}$$

Figure 2. The CSP of the program from Fig. 1

(CSP), (V, D, CO), is characterized by a set of variables $V = (v_1, ..., v_n)$, each variable having a domain D, and a set of constraints $CO = (c_1, ..c_k)$ which defines a relation R between variables. The variables in a relation $R \in CO$ are called the scope S_R of the relation.

Having the program given in Fig. 1 then its corresponding CSP is the given in Fig. 2

There are very effective reasoning algorithms available for CSP, e.g., for computing solutions. A solution of a CSP is an assignment of values to the CSP's variables which does not contradict any given constraint. State of the art constraint solvers are available for solving CSPs. More information about CSPs can be found in Rina Dechter's book on constraints [5].

It is possible, due to an incorrect build of the constraints system, for a CSP to become *inconsistent*. A *CSP is inconsistent* when there exists no assignment to its variables such that all constraints are simultaneously satisfied. There are several methods for testing the consistency of constraints system. The best known of them are arc consistency, path consistency and n-consistency check. A description of these methods is found in [5]. An optimized method for consistency check is the Max-Restricted Path Consistency method [6].

3 CSP Partitioning

When grouping functions into clusters the partitioning problem appears. For every cluster, the partitioning algorithm must assure that the quality criteria, e.g., time, bus load, together with the resource limitations, e.g., CPU, memory, are fulfilled with respect to the *control unit* (CU) where a cluster is executed. The CSP representation assures a natural way of depicting and combining all these requirements. When we build the CSP of the system we take into account the following types of constraints:

1. *Resource Constraints*: The resources of the CU, on which the cluster is executed, give us the resource constraints system. The memory of the CU and the processing power, are criteria which impose restrictions on the cluster that can be executed on the given CU. We cannot execute for example a cluster which needs 500Kb of memory on a CU which only has 300Kb of memory available.

2. *Quality Constraints*: Using quality functions we define the quality constraints. They assure that the system will behave within the given quality criteria. For example, if we want to have the bus load always under 50% then we have to define the quality functions such that this limit is never exceeded. From the quality functions we extract the quality constraints.

3. *Cost Constraints*: The cost constraints are given by the implementation's cost of the CUs. There can be more types of CUs with different properties and different implementation costs. It is possible that although a certain CU is expensive to implement it offers an all around smaller cost than when using 10 CUs that perform the same task. An optimal cost is hard to achieve. These types of constraints are strongly connected with an arbitrary parameter which we call *desired general cost* (DGC). We define the cost constraints such that they always assure that the 'all around system's costs' is smaller than the DGC. We also try to have the costs as low as possible without cutting off too much from the system's performance.

By combining these constraint systems we successfully build the CSP of the analyzed system. A solution to this CSP is a valid cluster partitioning of the system's function blocks.

Observation. It is possible, after combining the above constraints, that the resulted CSP is in an inconsistent state; that is a solution cannot be successfully computed. For example, if through the cost constraints system we specify that the DGC is k and through the resource constraints system we specify that we only have components that cost p, where $p < k$ and $p > 1$. We know we need at least k CUs so that the system can function correctly. Then we have the following constraints system: $(k * p < k) \wedge (p < k) \wedge (p > 1)$. It can be seen that these constraints system has no valid solution. In this situation we have to revise those constraints that can be adapted such that the inconsistent state is removed. In our example if we set the DGC level to $(p * k + 1)$ the constraints system leaves the inconsistent state.

3.1 Resource Parametrization

In order to build the system's CSP we first define the parameters that describe the system's behavior. There are two types of resources that we parameterize: the CUs and the functions that have to be executed by the system.

Figure 3. An abstract representation of the functions's network

We want to define the function blocks distribution for an automotive system over a set of available electronic control units (ECUs).

Within an automotive system there are different functions that have to be implemented. These functions have different safety levels. Some of the functions are safety critical, the ABS function, torque vectoring, or control of the attitude angle, and other have a lower importance degree, e.g., entertainment functions like DVD playing. All these function blocks are connected to each other by means of messages and data exchange mechanisms like bus protocols. Due to this, an automotive system can function correctly. Let's presume that we have t function blocks, $F = \{f_1...f_t\}$, that have to be executed on a minimal number of ECUs. We build the network functions as follows: the nodes of the network are the function blocks that have to be executed. Between functions that communicate there exists a connection in the network. Each connection has a label which denotes the communication frequency between the connected function blocks.

Let $CF = \{CF_1...CF_t\}$ be the set that denotes the communication frequency sets of the system's functions; e.g., $CF_i = \{cf_{i1}, ...cf_{it}\}$ is the set that describes the communication frequency of function f_i with respect to all other functions from the system. If there exists cf_{ij} in the set CF_i of a function f_i such that $cf_{ij} = 0$ then it means that there exist no network connection between function f_i and function f_j. A graphical depiction of such a network is given in Fig. 3.

The CF set helps us build the quality functions. The quality function receives as input-parameter the CF_i set of every function f_i, where $1 \leq i \leq t$. After we have built the network of functions we start the parameterizing process for the ECU's. They help us construct the Resource Constraints System. Let $ECU = \{ECU_1...ECU_k\}$ be the set of the available ECUs, then for each $ECU_i \in ECU$ such that $1 \leq i \leq k$. We define mem_i as being the available memory of module ECU_i and $proc_i$ as being the processing power of module ECU_i. Let the set $MEM = \{mem_1...mem_k\}$ be the set of ECU's memories and let $PROC = \{proc_1...proc_k\}$ be the set that describes the processing performances of the available ECUs. The sets MEM and $PROC$ represent the most simplified description of the available resources of an ECU. They suffice to describe how the resource constraints building process takes place. Each distributed system can have supplementary resources that have to be parameterized, but these basic resources are characteristics of every CU found on the market.

3.2 Building the CSP

In order to build the CSP of the system we have to build the three constraint systems: the resource constraints system, the quality constraints system and the cost constraints system. For this purpose we use the parameters introduced earlier: the memory and the computational power available on a given ECU, the memory and the computational power that a function block requires and the communication frequency that exists between functions.
We give the following formal definitions:

Definition 1 (Function Block) *Any function block (of t function blocks) is associated with a unique identifier f_i and its processing requirements $pow(f_i)$.*

Definition 2 (ECU) *Every Electronic Control Unit ECU_i is associated with a processing capacity max_{ECUpow_i}.*

We start building the *resource constraints system*. The following equations define the constraints system.

1. The overall memory consumption of the function blocks is smaller or equal to the available memory. Usually not all function blocks are executed at the same time, but in the worst case scenario, this trivial safety constraint assures us that no jamming occurs in the function execution process.
   $\sum_i mem(f_i) \leq \sum_j max_{ECUmem_j}$
2. An adjacent memory constraint is the *maximal function block memory constraint*. That is, let f_{max} be a function block such that the memory requirement of f_{max}, $mem_{f_{max}}$, is the maximum from all function's memory requirements. There exist an ECU, $ECU_k \in ECU$, with the available memory mem_k, such that $mem_k \geq mem_{f_{max}}$.
3. After we decide to deploy a cluster of functions, $C_j = \{f_i...f_{i+n}\}$,$i \geq 1$, on an ECU, ECU_j, then ECU_j must provide enough memory and processing power to host the deployed functional blocks. The function $deploy(ECU_j)$ returns the indices of the function blocks deployed on ECU_j.
   $\sum_{i \in deploy(ECU_j)}(mem(f_i) \leq max_{ECUmem_j}) \wedge$
   $\sum_{i \in deploy(ECU_j)}(pow(f_i) \leq max_{ECUpow_j})$
4. A function block is deployed on a single ECU only.
   $\forall i, j \in \{1..n\}, i \neq j \cdot deploy(ECU_j) \cap deploy(ECU_i) = \emptyset$
5. Any function $deploy$ that distributes all functional blocks f_i on max ECUs is a solution.
   $\{1..n\} = \bigcup_{j=1}^{max} deploy(ECU_j)$

By unifying the above constraints system we derive the resource constraints system (RCS):

$$RCS: \left\{ \begin{array}{c} 1.\sum_{\mathtt{i}} \mathtt{mem(f_i)} \leq \sum_{\mathtt{j}} \mathtt{max_{ECUmem_j}}; \\ 2.\exists f_i | f_i \in F, i \in [1,t], \forall j \in [1,t], \\ i \neq j, mem_{f_i} \geq mem_{f_j} \\ \Rightarrow \exists ECU_l \in ECU, l \in [1,k]: \\ mem(ECU_l) \geq mem_{f_i}; \\ 3.\sum_{i \in deploy(ECU_j)}(mem(f_i) \leq max_{ECUmem_j}) \wedge \\ \sum_{i \in deploy(ECU_j)}(pow(f_i) \leq max_{ECUpow_j}); \\ 4.\forall i,j \in \{1..n\}, i \neq j, deploy(ECU_j) \\ \cap deploy(ECU_i) = \emptyset; \\ 5.\{1..n\} = \bigcup_{j=1}^{max} deploy(ECU_j); \end{array} \right\}$$

The *quality constraints system* are the most important factor when we partition the function blocks into clusters. In order to build these constraints system we use a set of functions, named *quality functions*. The quality functions offer us a metric for computing the optimal partitioning of the function blocks. The constraints are created by imposing output values that these functions should not exceed for a given cluster. The constraint solver tries to find a set of function blocks such that all the quality constraints are fulfilled. When it finds such a set it creates the cluster.

Besides, as an extra quality constraint, we try to keep the output values of the quality functions to a level close to optimal (such that the cost is minimal). Each quality function receives as input parameter the CF set. How this set is built depends on the user and on the described system. There are more solutions proposed for building this set; one, given in [4], proposes a representation of the CF set by means of a geometrical matrix. It is beyond the scope of this paper to discuss how the CF is created. We presume that the set is already given and use it directly as input for the quality functions.

We build the quality constraints system based on the quality functions set. We use the quality functions presented in [4].

We define the following:

Definition 3 (Cluster's external cost) *It represents the frequency with which the function blocks within a cluster C_i, $i \in [1,c]$, communicate with the rest of the function blocks from the network. We denote this metric through E_i and we compute it as the average CF between the function blocks within the cluster and the external function blocks.*

Definition 4 (Cluster's internal costs) *It represents the frequency with which the function blocks communicate with each other within a given cluster C_i, $i \in [1,c]$. We denote this metric by I_i and it represents the average of all CF within the cluster C_i.*

Definition 5 (Cluster's diameter) *It represents, based on the CF of the function blocks, the average distance between the function within a given cluster C_i, $i \in [1,c]$. We denote this metric through $diamC_i$.*

Definition 6 (Distance between Clusters) *It represents, based on the CF of the function blocks, the average distance between a cluster C_i and a cluster C_j, $i,j \in [1,c]$, $i \neq j$. We denote this metric by $d(C_i, C_j)$.*

Definition 7 (External costs between clusters) *It represents, based on the CF of the function blocks, the external cost between a cluster C_i and the function blocks of a cluster C_j, $i,j \in [1,c]$, $i \neq j$. We denote this metric by $E(C_i, C_j)$.*

Definition 8 (Cluster's Nodes) *It represents the number of function blocks within a cluster C_i, $i \in [1,c]$. We denote this metric by N_i.*

The quality functions are defined below. Detailed informations about these functions can be found in [4].

1. *The External-Internal Ratio* is a ratio between the external and the internal costs must be as low as possible. That is, a good cluster is a cluster which communicates as little as possible with the other function blocks from the network and that has the internal communication frequency as high as possible. We define for every cluster a communication ratio limit, CRL_{max}, which represents the qualitative limit that every cluster must respect.
   $\forall C_i, i \in [1,c]\ \frac{E_i}{I_i} \leq CRL_{max}$

2. *The Davies Bouldin Criteria* shows a good partitioning when the factor is as low as possible. The Davies Bouldin (DB) factor is computed only after all the cluster are formed. We set a limit, DB_{max} that should never be surpass by the final cluster partitioning. After computing all the clusters c, we compute the DB factor. If it is greater than DB_{max} then the constraint is violated and a new partitioning of the function blocks is performed. If the constraint holds a valid configuration with respect to the DB factor was found.
   $$DB = \frac{1}{c}\sum_{i=1}^{c} max_{j \neq i}\left[\frac{diam(C_i) + diam(C_j)}{d(C_i, C_j)}\right]$$

3. *The Modularization Factor* (MF) is an indicator of a compact clustering of the function's blocks. The value of this factor should be as high as possible. For our constraints system we settle a minimal value, MF_{min}, below which the optimality criteria is violated. If, after computing all the clusters, we observe that the value of MF is smaller than MF_{min}, then the constraint is violated and we discard the partitioning. If the value of MF is greater than MF_{min} then we found a valid solution.
   $$MF = \frac{\sum_i I_i}{\sum_i \frac{N_i(N_i-1)}{2}} - \frac{\sum_{i<j} E(C_i, C_j)}{\sum_{i<j} N_i N_j}$$

4. *The SILHOUETTE factor* (Sh) verifies the correctness of the distribution of a function f_i within a cluster C_i with respect to a neighbor node C_j. The domain of the Sh value of the function f_i is $[-1, 1]$. A good distribution of the functions f_i within a cluster C_i, has the Sh value in the vicinity of 1. For every function f_i, we compute $Sh(f_i)$. If this value diverges with more than δ_{max} from 1 then the constraint is violated, the function is not distributed within cluster C_i and we start the search for a new cluster.
$Sh(f_i) = \frac{d(f_i,C_j)-d(f_i,C_i)}{max(d(f_i,C_j),d(f_i,C_i))}$

5. *The Cluster Load Deviation* (CLD) is computed after all the clusters c are created. Small values of this function denote a good partitioning of the function blocks. In a good case scenario all the clusters have a similar number of function blocks within them. We have the following constraint: the final CLD value of the network must not be greater than an optimal criteria CLD_{max}. If the CLD of the network is greater than CLD_{max} the partitioning of the function blocks is discarded and we restart the partitioning process. If the value of CLD is smaller than CLD_{max} then we have found a valid partitioning.
$$CLD = \sqrt{\frac{1}{c-1}\sum_{i=1}^{c}(N_i - \bar{N})^2}\ , \bar{N} = \frac{1}{c}\sum_{i=1}^{c}N_i$$

By combining the above criteria we build the Quality Constraints System (QCS). The CRL_{max}, DB_{max}, MF_{min}, δ_{max} and the CLD_{max} must be given by the user with respect to the desired system performances.

$$QCS: \left\{ \begin{array}{c} 1.\forall C_i, i \in [1,c]\ \frac{E_i}{I_i} \leq CRL_{max}; \\ 2.DB = \frac{1}{c}\sum_{i=1}^{c} max_{j\neq i}\left[\frac{diam(C_i)+diam(C_j)}{d(C_i,C_j)}\right] \wedge \\ (DB \leq DB_{max}) \\ 3.MF = \frac{\sum_i I_i}{\sum_i \frac{N_i(N_i-1)}{2}} - \frac{\sum_{i<j} E(C_i,C_j)}{\sum_{i<j} N_i N_j} \wedge \\ (MF \geq MF_{min}); \\ 4.Sh(f_i) = \frac{d(f_i,C_j)-d(f_i,C_i)}{max(d(f_i,C_j),d(f_i,C_i))} \wedge \\ ((1 - Sh(f_i)) \leq \delta_{max}); \\ 5.CLD = \sqrt{\frac{1}{c-1}\sum_{i=1}^{c}(N_i - \bar{N})^2} \wedge \\ (CLD \leq CLD_{max}); \end{array} \right\}$$

The *Cost Constraints System* (CCS) is built based on the system's cost criteria. Each ECU has a price and a performance description associated to it. We use the following constraints in order to build the CCS.

1. *The Price Constraint.* Given a network of function blocks F, a set of ECUs and a desired general cost DGC, then we have to distribute all the function set F over a number N_E of ECUs such that the total cost of these ECUs, P_{N_E} is smaller than DGC

2. The Bus Load Constraint (BLD) of the system must be lower than an imposed value, BLD_{max}. That is, we have to choose the ECUs on which we distribute the function blocks, such that the bus load of the system is never greater as the imposed value BLD_{max}.

By combining the RCS with the QCS and the CCS we derive the CSP associated to the system:
$CSP = RCS \cup QCS \cup CCS$

4 Constraint Solvers

A solution to a CSP is a valid assignment to all CSP's variables that does not violate any of the constraints from the CSP. For solving a CSP we use *CSP solvers*.A CSP solver is a software tool that, by means of different algorithms, tries to detect and remove the inconsistencies from a constraints system and to offer a valid solution to a given CSP. There are more state of the art CSP solvers which can offer good performance when solving a CSP. In this chapter we present four constraint solvers, CHOCO, JaCoP, MINION and TREE*. Each CSP solver has its weak points and its strong points and it depends on the size and type of the CSP which of the constraint solver is best fit for a given task. Because of that a comparison between the CSP solvers is not always possible.

We find a comparison and a description of ***JaCoP*** and ***CHOCO*** in [7]. The authors say that the all around performance of the JaCoP solver is better (54% faster) than of the CHOCO solver. However when talking about small CSPs the CHOCO solver is consireably faster than JaCoP. The two CSP solvers have similar specifications with regard to the type of variables and constraints that they can handle. Both constraint solvers offer a wide range of operations for describing constraints. Both tools are free of charge for academic porpuses. We find a description of the CHOCO solver in [10] and of JaCoP in [11].

The ***Minion CSP solver*** is fully presented in [8]. It allows four type of variable's domains and the input language supports definition of up to three dimensional matrices of decision variables. The set of primitive constraints is smaller than by CHOCO and JaCoP. However it includes the basic constraints like *equalities, inequalities, sum, product* and so on. The MINION project is an open source project and is still under development. A particularity of this CSP solver is the fact that variables representation is optimized at hardware level with respect to the solving algorithm (backtracking).

The ***TREE* CSP solver*** is presented in [9]. The TREE* solver is best suited for binary constraint systems. It can also work on integer variables but it takes a long time to compute a solution. That makes it not a valid choice for big CSPs. It implements the basic operations needed for

creating constraints. A particularity of this CSP solver is that it does not use backtracking in order to get a solution, but simulates instead, by means of tables, all the possible solutions of the constraints from the CSP. It then joins all the constraints and the result is a valid CSP-solution. The TREE* solver is an open source product and still under development.

As our model contains arithmetic as well as logical operators and we are required to provide a rather detailed model, JaCoP and Minion CSP solvers appear to be a reasonable choice for our specific task.

5 Conclusions and related work

Constraints are a natural way of representing complex problems and are often used in the area of configuration and reconfiguration of diverse systems. Constraint Satisfaction Problem (CSP) representations are successfully used in diverse areas from software engineering like configuration and reconfiguration of large systems [13], recommender systems like CAWICOMS which is presented in [15], and software task planning [7].

In this article we outline a novel modeling approach that allows for deployment of embedded automotive software. Our purely model-based approach allows for fully automatic deployment of software functions in a resource-constrained software system. For ease of discussion, we exemplify this for general constraints like memory consumption and bus load, however, our approach can be extended in a natural vein.

Besides of its applicability in an early stage of development, most notably, our model incorporates well-known quality criteria from algorithmic software deployment approaches.

In addition to the straightforward and natural problem representation our model allows for computation of a valid solution satisfying the outlined criteria (resource contraints, quality constraints, cost constraints ...) by relying on standard CSP solvers. Moreover we do not have to generate all the cluster combinations but rely on the the first n, $n \geq 1$, solutions that the CSP solver comes up with.

References

[1] Henrich Druck & Medien GmbH, Challenges for the automotive supply chain, *Association of German Car Manufacturers (VDA) HAWK2015*, Frankfurt am Main, 2003.

[2] E. Schoitsch, Design for Safety AND Security of Complex Embedded Systems: A Unified Approach, *NATO Advanced Research Workshops, Cyberspace Security and Defense: Research Issues*, p. 161-174, Springer Dordrecht, Berlin, Heidelberg, New York, 2004.

[3] R. Henia, A. Hamann, M. Jersak, Razvan Racu, Kai Richter, Rolf Ernst, System Level Performance Analysis - the SymTA/S Approach, *IEEE Proceedings Computers and Digital Techniques*, 152(2):148–166, March 2005.

[4] S. Brummund, N. Kehl, P. Nenninger and U. Kiencke, ISODATA Clustering for Optimized Software Allocation in Distributed Automotive Electronic Systems, *SAE World Congress & Exhibition*, Detroit, MI, USA, Session: In-Vehicle Networks, 2006.

[5] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[6] R. Debruyne and C. Bessiere, From Restricted Path Consistency to Max-Restricted Path Consistency, *Principles and Practice of Constraint Programming*, Berlin/Heidelberg, pp. 312-326, 1997.

[7] D.Benavides, S. Segura, P. Trinidad and A. Ruiz-Cortes, Using Java CSP Solvers in the Automated Analysis of the Feature Models *GTTSE*, Braga, Portugal, pp. 399-408, 2006.

[8] I. P. Gent, C. Jefferson and I. Miguel, Minion: A Fast, Scalable, Constraints Solver, *ECAI*Riva del Garda, Italy, 2006.

[9] M. Stumptner and F. Wotawa, Coupling CSP decomposition methods and diagnosis algorithms for tree-structured systems, In *Proc. 18th International Joint Conf. on Artificial Intelligence*, pages 388–393, Acapulco, Mexico, 2003.

[10] F. Laburthe and N. Jussien, CHOCO constraint programming system, *http://choco.sourceforge.net*, 2003-2006.

[11] K. Kuchcinski, Constraints-driven scheduling and resource assignment, *ACM Transaction on Desgin Atumation of Electronic Systems (TODAES)* , 8(3):355-383, July 2003.

[12] L. Ardissono, A. Felfernig, G. Friedrich, D. Jannach, R. Schafer, M. Zanker, A Framework for Rapid Development of Advanced Web-based Configurator Applications, *AI Magazine* , 24(3), 93-110, 2003.

[13] G. Fleischanderl, G. E. Friedrich, A. Haselbck,H. Schreiner, M. Stumptner, Configuring Large Systems Using Generative Constraint Satisfaction IEEE Intelligent Systems, Volume 13, Issue 4, pp. 59 - 68, 1998.

Applying Critical Pair Analysis in Graph Transformation Systems to Detect Syntactic Aspect Interaction in UML State Diagrams

Zaid Altahat[1,2] , ***Tzilla Elrad***[1] , *and* ***Luay Tahat***[1]
[1] *Illinois Institute of Technology (IIT)*
Chicago, IL. USA
{altazai, elrad, tahaway}@iit.edu
[2] *GE Healthcare*
zaid.altahat@ge.com

Abstract

Aspect Oriented Modeling (AOM) separates crosscutting concerns by defining Aspects and composition mechanisms at the model level. Composition of multiple Aspects will most likely result in more than one Aspect matching the same join points. This will create interaction among Aspects, in the sense that one Aspect will enable or disable another Aspect. Aspect Interaction is similar to a phenomenon that exists in graph transformation systems where multiple Graph Transformation (GT) rules share some conflicting elements, it is referred to as Critical Pair Analysis (CPA) and it provides an algebraic-based mechanism to detect and analyze the interaction of the rules. In our proposed framework the same mechanism is used to analyze and detect Aspect Interaction in UML State Diagrams. It achieves that by performing a model-transformation of the Aspects to the graph transformation rules. This will enable developers to specify only the order precedence for interacting Aspects rather than for the combinations of every Aspect to every other Aspect in the system. The interaction detection is modular, independent of the base model; this adds the advantage of not having to re-evaluate the interaction each time the base model changes.

Keywords

Aspect Oriented Software Development; Aspect Oriented Modeling; Aspect Interaction; Critical Pair Analysis.

1. Introduction

Software modules are added to other software modules and other components in an incremental way to build software products. This process will most probably result in interaction among the software modules. Most software modules have several complex interactions with other software module through their life cycle. This interaction was studied earlier in the telephony systems and referred to as Feature Interaction (FI) [3, 6, 10, 13]. Different mechanisms [6, 10, 13, 34] were proposed to handle the FI problem. Aspect Oriented Software Development (AOSD) [35] builds software systems by composing crosscutting concerns in a similar approach to the features in the telephony systems. This leads to the Aspect Interaction (AI) problem that is very similar to the FI problem. The AI is not necessarily harmful [25]. But the term AI usually refers to the unintended interaction. If the interaction is planned, order precedence needs to be defined [20]. If a dependency between two Aspects is not planned, then unless an AI detection mechanism is used, the dependency might slip undetected with potential harm to the system. The Motorola WEAVR [17] has reported the AI problem in the Telecomm industry, where precedence is defined for interacting Aspects [20]. Graph Transformation (GT) systems have developed a mechanism to detect conflicts among GT rules [38].

GT rules are used to apply changes to a host graph. Two GT rules that overlap are said to be critical pair. Critical Pair Analysis (CPA) is used to detect conflicts and dependencies in GT Systems. Two rules are in conflict if one rule disables another rule. On the other hand, two rules are dependent if one rule enables the other.

Since applying Aspects to a base model involves matching and modifying elements in the base model similar to those of the GT systems, CPA is used to detect syntactic interaction among aspects in the UML State Diagrams. UML State Diagrams are increasingly used in modeling wide range of embedded devices, from small gadgets to Telecom Systems [17], where Specification and Description Language (SDL) [39] is used as an Action Language to generate code from models.

We are proposing a graph-based framework for the detection of the unintended interaction among Aspects. Using the proposed approach, users will be

able to identify interacting Aspects independently of the base model. This way users need to define only order precedence for identified Aspects once, when they are defined, not each time the model changes.

This paper is organized as follows; Section 2 explains the different types of AI. Section 3 relates the GT systems and CPA to the Aspects and AI. Section 4 presents a case study by applying the framework to an example of an ATM modeled as State Diagram. In the case study multiple Aspects are defined and their interactions are analyzed and classified using the generated CPA report. Related work is discussed in section 5. Conclusion and future work are discussed in section 6.

2. Aspect Interaction

With the use of AOSD to manage separation of concerns, AI is an inevitable issue. AI takes place when multiple Aspects share conflicting elements in their pointcuts or advices. Multiple aspects are said to be independent if the order of applying aspects result in the same model. Two models are considered syntactically the same if there is a bijective mapping between the two models. That is given two models M1 and M2, for each element in M1 there is one element in M2 with the same properties. There is also the same reverse mapping from M2 to M1.

Interaction among Aspects exists in the form of either dependency or conflict. This kind of AI is referred to as Aspect-to-Aspect interaction [20]. Even non-conflicting and independent Aspects might have unintended impact on the structure of the base model; this kind of interaction is referred to as Aspect-Base interaction. Aspects may also have unintended impact on the behavior of the base model, this kind of interaction is referred to as *semantical* interaction [5]. Currently the proposed framework studies the Aspect-to-Aspect interaction, the other types of interactions are planned for future work.

Without the AI analysis, Aspects designers would have to specify order precedence for all Aspects in the system. This requires large efforts part of which are useless and wasted. Aspects' designers may still be interested in specifying order for certain Aspects, but they do not have to specify it for all Aspects. Using CPA in the analysis also adds the advantage of having AI analysis independent of the base model. So changes to the base model will not affect the AI, only changing Aspects will require a re-run of the analysis.

The next 4 definitions will shed light on the different types of interactions. Let:

$\mathbf{M_1}$ = The result of applying Aspect A1 to the Base Model (BM).
$\mathbf{M_2}$ = The result of applying Aspect A2 to the BM.
$\mathbf{M_{12}}$ = The result of applying A1 *then* A2 to the BM.
$\mathbf{M_{21}}$ = The result of applying A2 *then* A1 to the BM.

Definition 1: Two Aspects do not have interaction between them iff $\mathbf{M_{12}} = \mathbf{M_{21}}$.

Definition 2: A *dependency* exists between two Aspects if ($\mathbf{M_{12}} = \mathbf{M_2}$) or ($\mathbf{M_{21}}=\mathbf{M_1}$)

Definition 3: A *conflict* exists between two aspects if ($\mathbf{M_{12}} = \mathbf{M_1}$) or ($\mathbf{M_{21}} = \mathbf{M_2}$)

Definition 1 states that, regardless of the order of applying Aspects, the output model is the same. This is only possible if the application of one Aspect does not alter the applicability of the other Aspect. If $\mathbf{M_{12}} \neq \mathbf{M_{21}}$, then AI exists between A1 and A2 in the form of either dependency (definition 2) or conflict (definition 3). If ($\mathbf{M_{12}} \neq \mathbf{M_{21}}$ *And* $\mathbf{M_{12}}=\mathbf{M_2}$) then A1 depends on A2, or if ($\mathbf{M_{12}} \neq \mathbf{M_{21}}$ *And* $\mathbf{M_{21}}=\mathbf{M_1}$) then A2 depends on A1. A conflict is defined as either ($\mathbf{M_{12}} \neq \mathbf{M_{21}}$ *And* $\mathbf{M_{12}}=\mathbf{M_1}$) or ($\mathbf{M_{12}} \neq \mathbf{M_{21}}$ *And* $\mathbf{M_{21}}=\mathbf{M_2}$), which means A1 disables A2, or A2 disables A1, respectively.

The proposed framework will detect potential conflicts and dependencies among Aspects without the need to check the base model for the pointcuts applicability. It achieves this by inspecting all combinations of the pointcuts and advices of all Aspects in a pair-wise manner. This approach will report potential interaction among all Aspects, even if some Aspects might not have a match. The advantage of this approach is to avoid regenerating the AI report each time the base model changes. The second approach is to eliminate some Aspects from the analysis if they do not have a match. This will result in fewer Aspects to analyze but the report will need to be regenerated each time the base model changes. Also in the second approach the base model needs to be entirely transformed to a graph, which will add a considerable overhead to the analysis.

3. Critical Pair Analysis and Aspects

This section describes CPA, graph transformation, and their relation to Aspects defined in UML State Diagrams. A graph transformation applies a GT rule (**P** = **L, R**) to a host graph G; where **P** is a production, **L** is the left hand side (LHS) graph, and **R** is the right hand side (RHS) graph. **P** may also have a set of Negative Application Conditions (NAC), which are elements that may not exist for a rule to apply. A GT rule replaces graph **L** with **R** in host graph G.

Critical Pair Analysis (CPA) [23] is "a pair of transformations both starting at a common graph G such that both transformations are in conflict, and graph G is minimal according to the rules applied." [32]. That is the GT rules **P1** and **P2** form a critical pair if both, **P1** and **P2**, can be applied to the same minimal graph G. But applying **P1** will prohibits the application of **P2** and/or vice versa. Certain tools, such as Attributed Graph Grammar (AGG) [1], provide graph transformations and CPA. An attributed graph allows the definition of attributes on graph elements. CPA and NAC [38] are combined to detect conflicts in GT systems, similar approach is used in the proposed framework for Aspects.

Figure 1 shows a simple graph, referred to as host graph, (left), a GT rule (middle) with its LHS (**L**) and its RHS (**R**) components, and the generated graph (right). Host graph is searched for a graph morphism of ***L***, referred to as a *match*. A graph morphism between two graphs, G and H, is a bijective mapping (ψ) between the vertices of G and the vertices of H, such that two vertices u and v are adjacent in G iff their mapping vertices $\psi(u)$ and $\psi(v)$ are adjacent in H. If a match is found, the graph ***R*** is applied to the host graph. GT works as follows:

- Elements in *L* and in *R* are preserved in the generated graph.
- Elements in *L* but not in *R* are deleted from the generated graph.
- Elements in *R* but not in *L* are created in the generated graph.

Figure 1-(middle) shows a NAC edge between the states '**b**' and '**c**' in the LHS, marked with '**X**'. The NAC will be used in transforming some of the pointcut constructs, such as 'XOR'. With out the NAC edge in Figure 1, the matching mechanism will only check for the existence of vertices '**b'** and '**c'** without checking the absence, or presence, of an edge between '**b**' and '**c**'. These requirements for morphism come from L. According to **R** edge '**e4**' is created and vertex '**d**' is removed. Generated graph is presented in Figure 1 (right).

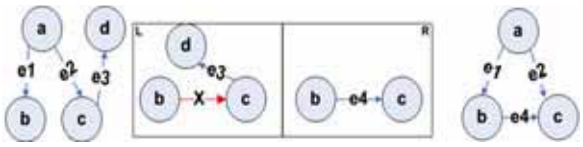


Figure 1 A graph transformation rule on a directed labeled host graph.

Aspect Oriented Modeling (AOM) [19] follows an approach similar to the GT systems by querying and adapting base model elements. If a mapping is created between GT rules and Aspects in AOM, then the CPA technique can be used to analyze AI to detect any conflicts and dependencies among Aspects. The proposed framework model-transforms Aspects into some GT rules and run AGG on the generated rules to produce a CPA report.

4. An ATM Case Study

In this section we present an example that will demonstrate how Aspects with pointcuts consist of composite state and compound transitions are transformed to some GT rules. The example consists of an ATM machine described by the UML State Diagram presented in Figure 2. The ATM lacks the behavior to diagnose and early terminate the ATM machine. The behavior is added to the base ATM model by the 'diagnostic' concern, presented in section 4.2, that has 4 Aspects.

4.1. The ATM State Diagram

Figure 2 presents the UML State Diagram for a bank ATM. Since the *Active* state is composite non-orthogonal, only state '*validating'* will have the incoming transition '*card_in*'. The *Maintenance* state is orthogonal, so both states '*testing'* and '*waiting'* will receive the incoming transitions '*maintain'*.

In order to make it easier on the reader to follow, we numbered each state in Figure 2 and used the numbers in the generated GT rules. Vertices whose names are separated by a '|', for instance the vertex '**1|4**', represent substates in the composite orthogonal state *Maintenance*. Digits to the left of '|' come from the top region, and digits to the right come from the bottom region. When the state '*Maintenance'* becomes active, states '*testing' **(1)*** and '*waiting' **(4)*** become active.

4.2. Transformation of Pointcuts to Graph Transformation Rules

The following 4 Figures, 3 through 6, present the 4 Aspects which are part of the concern 'diagnostic' that will add the behavior to diagnose and early-terminate the ATM. Figure 3-(a) presents the first Aspect A1. Elements marked with '**E**' are exposed and passed to the weaver to adapt. Also to simplify presentation of the GT rules, if an element is presented in the LHS but not in the RHS, it does not mean that the element is deleted, they are just not shown for simplicity.

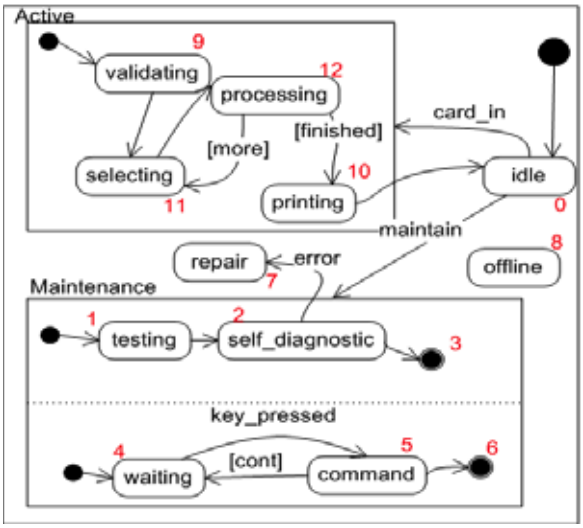


Figure 2 A UML State Diagram of an ATM

The Aspect in Figure 3-a is transformed to three different GT rules shown in Figure 3-b. The three vertices (2|6), (2|5), and (2|4) represent the different states the composite state 'Maintenance' might be in while in state 'self_diagnostic' (2). The RHS of the GT rules in Figure 3-b represent the creation of the edge 'diagnostic' between the states 'validating' (9) and 'self_diagnostic' (2).

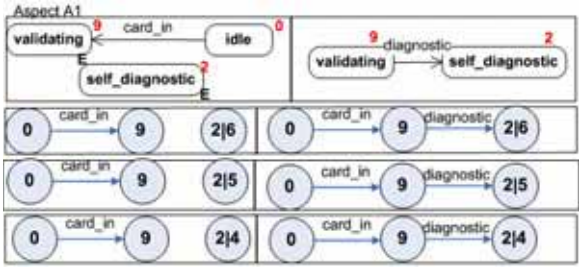


Figure 3-a(top) Aspect 1, b-(bottom) GT rules

Figure 4 presents the second Aspect A2. The advice of the Aspect creates the edge 'eject' to the sequential state 'Active'. Every substate in the state 'Active' will have an incoming edge labeled 'eject' from the state 'idle'. The vertices 9,10,11, and 12 of the generated GT rules in Figure 4-b represent the substates of state 'Active'. Note, to simplify the presentation of the GT rules, the RHSs do not show the edge 'diagnostic' and the vertices (2|6), (2|5), and (2|4) which are preserved in the host graph.

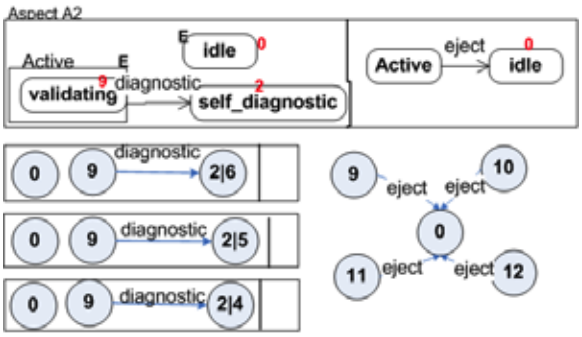


Figure 4 a-(top) Aspect 2, b-(bottom) GT rules

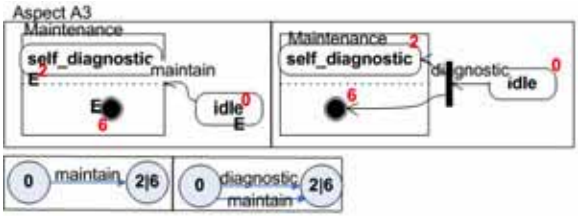


Figure 5 a-(top) Aspect 3, b-(bottom) GT rules

Figure 5 presents the third Aspect A3. The Aspects creates the *fork* transition 'diagnostic' which forks to the two substates, 'self_diagnostic' and the final state of the bottom region in the composite state 'Maintenance'. This results in one GT rule shown in Figure 5-b.

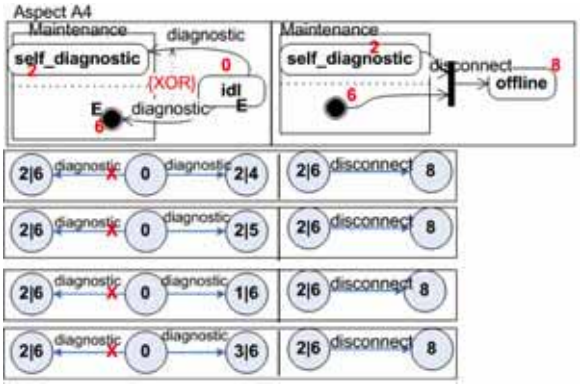


Figure 6 a-(top) Aspect 4, b-(bottom) 4GT rules

Figure 6 presents the fourth Aspect A4. The pointcut will match the edge 'diagnostic' from the state 'idle' to the state 'self_diagnostic', or from the state 'idle' to the state final state of the bottom region, but not from both. The NAC is used to transform the 'XOR' element. First two of the 4 GT rules in Figure 6-b present the GT rules that match the transition 'diagnostic', and the states 'idle' and 'self_diagnostic'. At the same time it does not allow the same transition between the states 'idle' and the *final* substate. The bottom 2 GT rules show the opposite.

The GT rules of the four Aspects presented in Figures 3-b, 4-b, 5-b, and 6-b are fed to the AGG to generate the CPA report. There are a total of 11 GT rules. To trace the GT rules back to the Aspects, each of the GT rule's name consist of two parts separated by a hyphen. For example the GT rule "A1-R1" represents the first GT rule (R1) of the Aspect A1, presented in Figure 3. The GT set "A1-*" refers to all the GT rules in A1. Any pair of rules in the GT set "A1-*" that is in conflict or dependency with any GT rule in other GT sets, will cause the Aspect A1 to be in the same conflict or dependency as its rule. For instance there is a conflict, Figure 7 (top), between the rules A3-R1 and A4-R1, which causes the Aspects A3, and A4 to be in conflict. This is because the pointcut of A4 doesn't allow for the transitions created by A3 for its pointcut to have a match. Also by inspecting Figure 7 bottom, we can see that A2 depends on A1 for its pointcut to find a match. One thing to mention is that any conflict or dependency within the same GT set is irrelevant and ignored. Note, due to space, Figure 7 shows only part of the report for the interacting Aspects.

Figure 7 The CPA report of the ATM statemachine.

5. Related Work

Several approaches are proposed to deal with Aspect Interaction at different phases of software design. First workshop on AI [26] was dedicated to this issue. For instance [27, 28, 29] study requirement interactions, [30] studies interaction in design models, and [31] gives good summary of negative impacts of Aspects on base program.

In [8] Aspect interaction is studied for Aspect-Oriented Programming (AOP) environment such as AspectJ. Aspects are classified based on their interaction to orthogonal (independent), unidirectional (an Aspect depends on another), or circular dependency between two Aspects. The AI in AOP is also classified into different types in [36, 37].

When AOSD was first introduced, [14] studied the interaction problem and proposed a framework for detecting aspect interactions at the language level for AOP. They are considered among the first to look at this problem. Order Precedence for the Aspect-to-Aspect interference of models in the Motorola WEAVR [17] was proposed by [20]. They define 3 precedence relations as *follows*, *hidden_by*, and *depdendent_on*. The authors' intent is not to detect interaction, but rather to define precedence relations for interacting Aspects. In [5] semantic conflicts between aspects and base model are studied. Authors translate models to Alloy [2] to be formally verified. Their approach is for semantic verification of aspects and base model interaction. For each aspect they define constrains, pre and post conditions, that will be verified using Alloy at the weaving time. Live Sequence Charts are used by [33] to detect AI at the Joinpoint in the form of use-case scenarios.

A graph-based approach [18] is used to detect composition conflicts due to weaving multiple aspects in AspectJ [22]. GROOVE [21] is used as the graph-rewriting tool. In their approach they detect pre-defined language violations, such as multiple conflicting method definitions, and cyclic inheritance. The essence of their work is to verify predefined rules in AspectJ, contrary to our approach, which is to detect conflicts among aspects.

An analysis of aspect interaction in AOP is provided in [7] that was applied to AspectJ. They provide a solution that is constraint-based and declarative for interacting aspects. Nevertheless, their work doesn't discuss mechanisms for detecting interaction among aspects. Our work concentrates on detecting interaction, dependency and conflicts, of aspects. A mechanism for semantic aspect interaction in Composition Filters for AOP is studied in [11]. Authors provide a mechanism similar to the mechanisms for detecting deadlock in a computer system. Based on the semantics of the added advices, their approach tries to order aspects in a harmless way.

The next two references [9, 12] are for UML models and are graph-based. CPA is used in [9] to analyze aspect interaction in UML class diagrams. Model transformations are expressed as pre and post conditions that are used in defining graph transformations rules. Pre and post conditions are derived from activity diagrams. In their approach classes and associations among classes are tracked using AGG to analyze their interaction. Creation and deletion of classes and their associations are mapped to graph transformation and further analyzed using AGG. CPA is used in [12] to detect feature

interaction in Software Production Lines (SPL). The paper presents a graph-based Modeling Aspects using a Transformation Approach (MATA) to specify how features, modeled in UML, relate to each other. Our framework is also graph based but for UML State Diagrams, in particular composite states and compound transitions and their transformation to GT rules. Our framework uses CPA technique to detect Aspect-to-Aspect Interaction.

6. Conclusion and Future work

We demonstrated how to detect AI in UML State Diagrams. The proposed framework uses Critical Pair Analysis in the GT Systems to detect the interaction, CPA is provided by AGG. The framework has a complexity of $O(n^2)$, where "n" is number of Aspects; but the AI detection for a pair of Aspects needs to be done only once in the system's lifetime. Hence the introduction of a new Aspect to the system will result in (n) pairs between the new and existing Aspects, AI among existing Aspects doesn't need to be reevaluated. Consequently, only a O(n) is needed for the introduction of a new Aspect. The proposed approach is modular (independent of the base model). This adds a huge advantage in large industrial system.

To be able to use CPA, Aspects are transformed to GT rules. KerMeta was used to execute all the model transformations. As seen in section 4, users do not have to define order precedence for all possible combinations of Aspects. Instead user is required only to define order between the Aspect A1 and A2 and precedence between A3 and A4.

However, the proposed framework does not support pattern matching in defining pointcuts, similar to those supported by AspectJ. This is due to the limitation enforced by AGG. There are also other mechanisms that are more expressive, such as Join Point Designation Diagram (JPDD) [4, 15] and the State Machine Joinpoint Model [16] used in the WEAVR [17]. Such mechanisms will result in different GT rules when integrated into our framework. In future work we plan on adding support for JPDD in our framework.

As seen in section 4 traceability between the GT rules and Aspects was done manually by using the Aspect#-Rule# naming convention. In large-scale production an automatic traceability is needed which will automatically decide which Aspects are in conflict or dependency without having to report the triggering GT rules.

7. References

1. AGG Homepage. http://tfs.cs.tu-berlin.de/agg/
2. Alloy Homepage. http://alloy.mit.edu.
3. D. Cansell, D. Méry. Abstraction And Refinement Of Features. Language Constructs for Designing Features. Springer Verlag, 2000.
4. D. Stein, S. Hanenberg, and R. Unland. "Expressing Different Conceptual Models Join Point Selections in Aspect-Oriented Design. Proceedings of the 5th int'l conference on AOSD. Bonn, Germany. ACM. 2006.
5. F. Mostefaoui, J. Vachon. "Design-Level Detection of Interactions in Aspect-UML Models Using Alloy". JOT. Aspect-Oriented Modeling Vol.6, No. 7. August 2007.
6. F. Beltagui. "Features and Aspects: Exploring feature-oriented and aspect-oriented programming interactions". Lancaster University. 2003.
7. I. Nagy, L. Bergmans and M. Aksit, "Composing Aspects at Shared Joinpoints", In Proceedings of International Conference NetObjectDays (NODe), Erfurt, Germany, pp. 19-38. September 2005.
8. J. Kienzle, Y. Yu, J. Xiong, "On Composition and Reuse of Aspects". In Proceedings of the 2nd Workshop on Foundations of Aspect-Oriented Languages (FOAL), Boston, MA, March 2003.
9. K. Mehner, M. Monga, G. Taentzer. "Interaction Analysis in Aspect-Oriented Models". 14th IEEE Int'l Req' Engineering Conference (RE'06) pp. 69-78. IEEE. 2006.
10. P. Zave. "Feature Interactions and Formal Specifications in Telecommunications". Vol 26, Issue 8. IEEE Comp Society Press Los Alamitos, CA, USA. 1993.
11. P. Durr, T. Staijen, L. Bergmans, and M. Aksit, "Reasoning About Semantic Conflicts Between Aspects." 2nd European Interactive Workshop on Aspects in Software (EIWAS), Brussels, Belgium, September 2005.
12. P. Jayaraman, J. Whittle, A. M. Elkhodary, and H. Gomaa. Model Composition in Product Lines and Feature Interaction Detection Using Critical Pair Analysis. Springer Berlin, Volume 4735/2007. 2007.
13. R. J. Hall. "Feature Combination and Interaction Detection via Foreground/Background Models". Computer Networks: The Int'l Journal of Computer and Telecom Networking. Volume 32, Issue 4. Elsevier North-Holland, Inc. New York, NY, USA. April 2000.
14. R. Douence, P. Fradet, M. Südholt "A Framework for the Detection and Resolution of Aspect Interactions." Vol2487/2002. Springer Berlin/ Hedelberg. 2002.
15. S. Hanenberg, D. Stein, R. Unland. "From Aspect-Oriented Design to Aspect-Oriented Programs: tool-supported translation of JPDDs into Code." Sixth Int'l Conference on AOSD. Vancouver, British Columbia. 2007.
16. T. Cottenier, A. van den Berg, T. Elrad. Joinpoint Inference from Behavioral Specification to Implementation. ECOOP. 2007.
17. T. Cottenier, A. van den Berg, T. Elrad. "The Motorola WEAVR: Model Weaving in a Large Industrial Context." Sixth Int'l Conference on Aspect-Oriented Software Development. Vancouver, British Columbia. 2007.

18. W. Havinga, I. Nagy, L. Bergmans, M. Aksit. “A graph-based approach to modeling and detecting composition conflicts related to introductions.” AOSD; Vol. 208. Proceedings of the 6th in’l conference on AOSD. Vancouver, British Columbia, Canada 2007.
19. www.aspect-modeling.org/
20. J. Zhang. T. Cottenier, , A. van den Berg, J. Gray. **“**Aspect Composition in the Motorola Aspect-Oriented Modeling Weaver”. JOT. Special Issue: Aspect-Oriented Modeling Vol.6, No. 7. August 2007.
21. http://groove.sourceforge.net/groove-index.html
22. http://www.eclipse.org/aspectj/
23. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. “Fundamentals of Algebraic Graph Transformation.” EATCS Monographs in TCS. Springer, 2005.
24. KerMeta Homepage: http://www.kermeta.org/
25. F. Sanen, E. Truyen, W. Joosen, A. Jackson, A. Nedos, S. Clarke, N. Loughran, A. Rashid. “Classifying and Documenting Aspect Interactions”. Workshop on Early Aspects at AOSD 2006
26. R. Chitchyan, J. Fabry, L. Bergmans, "Aspects, Dependencies, and Interactions", in Aspect, Dependencies, and Interactions Workshop (held at ECOOP). 2006.
27. S. Khan and A. Rashid, "Analyzing Requirements Dependencies and Change Impact Using Concern Slicing", in Aspects, Dependencies, and Interactions Workshop (held at ECOOP), Lancaster University Computing Department Technical Report Series, ISSN 1477447X. Lancaster, 2006.
28. J. Magno and A. Moreira, "Concern Interactions and Tradeoffs: Preparing Requirements to Architecture", in Aspects, Dependencies, and Interactions Workshop (held at ECOOP), Lancaster University Computing Department Technical Report Series, ISSN 1477447X. Lancaster, 2006.
29. D. Bar-On, and S. Tyszberowicz, "Derived Requirements Generation" in Aspects, Dependencies, and Interactions Workshop (held at ECOOP), Germany, 2007.
30. P. Shaker and D. K. Peters, "Design-level Detection of Interactions in Aspect-Oriented Systems", in Aspects, Dependencies, and Interactions Workshop (held at ECOOP), Lancaster University Computing Department Technical Report Series, ISSN 1477447X. Lancaster, 2006.
31. F. Munoz, O. Barais, and B. Baudry, "Vigilant Usage Of Aspects" in Aspects, Dependencies, and Interactions Workshop (held at ECOOP), Germany, 2007.
32. G. Taentzer, “AGG: A Graph Transformation Environment for Modeling and Validation of Software”, AGTIVE, US 2003.
33. S. Bakre, T. Elrad. ”Scenario Based Resolution of Aspect Interactions With Aspect Interaction Charts” in the 10th Int’l workshop on AOM (held at AOSD), Vancouver, Canada, 2007.
34. L. Tahat, B. Vaysburg, and B. Korel. “Requirement Based Automating Black-Box Test Generation.” Proc’s of the 25th Annual IEEE Int’l Computer Software and Applications Conference, 2001.
35. http://aosd.net/
36. M. Rinard, A. Salcianu, and S. Bugrara, “A Classification System and Analysis for AOP”. In Proc. 12th Int. FSE-12, Newport Beach, USA, Nov. 2004.
37. F. Tessier, M. Badri, and L. Badri, “A Model-Based Detection of Conflicts Between Crosscutting Concerns: Towards a Formal Approach”. Int. WAOSD, China, 2004.
38. L. Lambers, H. Ehrig, F. Orejas. “Conflict Detection for Graph Transformation with Negative Application Conditions”. Lecture Notes In Computer Science. Germany, 2006.
39. T. Cottenier, A. van den Berg., T. Elrad. Model Weaving: Bridging the Divide between Translationists and Elaborationists. Workshop on Aspect-Oriented Modeling at the 9th International Conference on Model Driven Engineering Languages and Systems, Milan, Italy, 2006.

Model Comparison: A Strategy-Based Approach

Kleinner Oliveira, Toacy Oliveira
Informatics Faculty
Pontifical Catholic University of Rio Grande do Sul
Ipiranga Avenue 6681 - Building 32 - ZIP 90619-900
Porto Alegre - Brazil
{ksoliveira,toacy}@inf.pucrs.br

***Abstract*— With the emergence of Model Driven Architecture (MDA), the role of model composition has become very important. One challenge of model composition is specifically to merge models expressed in the Unified Model Language (UML) and its profiles. However, for merging it is necessary to perform an essential task: *model comparison*. In this paper, we present a model comparison technique that relies on match strategies so that input models can be merged if they are considered equivalent according to a specific *match strategy*. To put this in practice we defined a *match operator* that makes use of match rules, synonym dictionary and typographic similarity. Moreover, a guidance for model comparison was elaborated to specify the activities that go along with model comparison.**

I. Introduction

A significant factor behind the difficulty of developing complex software is the wide conceptual gap between the problem and the domains of discourse [3], [4]. The model-driven approaches move development focus from third generation programming language code (e.g. Java code) to models, specifically models expressed in the Unified Model Language (UML) and its profiles [14], [16]. The goal is to manage the software at the level of its concepts in order to reduce the gap, quickly attain code and become the software development less difficult and costly. One reference to these approaches is the Model Driven Architecture (MDA) [10], an approach to Model Driven Development (MDD) from Object Management Group (OMG).

A typical MDA process involves a number of UML models to graphically represent a system's structure and behavior often defined in different platforms (such as J2EE or .NET) or domains (such as real-time or business process modeling) from a specific viewpoint and at a certain abstraction level that can be ultimately converted into the actual code by a model transformation engine. It can use models not only horizontally to describe different system aspects but also vertically, in order to be refined from higher to lower abstraction levels. Thus, the model-driven approaches make use of model transformation and model composition techniques to manipulate and manage UML models at the same and different abstraction levels. Models can represent concepts related to the system domain such as Telecom and Insurance, and also exposes the underlying execution infra-structure such as .NET or Java, which means a typical system can be represented by several models that must be somehow assembled (composed) into a cohesive unit.

The model composition can be viewed as an operation where a set of activities should be performed to merge two input models, M_A (receiving) and M_B (merged), in order to produce an output model, M_{AB}. In short, we can represent it by the equation: M_A+$M_B \rightarrow M_{AB}$. However, an important step to achieve model composition lays in the ability to compare input model elements, thus before merging M_A and M_B, it is necessary to compare to verify semantic and syntactic overlap in such models. The need to avoid such overlaps stands for the fact that the ultimate system's model should represent each concept uniquely to avoid conflicts, misinterpretation and mistransformation. For example, according to UML metamodel specification should not exist two (or more) models (e.g., two UML classes) with equal names in a same namespace, then a model composition mechanism should take in account such conditions to produce the output model, otherwise it can have conflicting names and elements with same semantic value.

In this paper we demonstrate the role and the importance of model comparison in model composition, describe the challenges that should be tackled to compare models and propose a *match operator* that is responsible for putting in practice a *strategy-based model comparison approach*. Moreover, a brief guidance for model comparison is exposed in order to specify the activities that go along with model comparison.

A. Motivating Example

We motivate our work with a composition example of two UML profiles, *Tree* and *Topology* [2] (see Figure 1) each representing a Domain-Specific Modeling Language (DSML). We have chosen UML profiles because they play a central role in the OMG's MDA approach. The Tree profile represents a common hierarchical data structure used for many computer science applications, while the Topology profile represents the connections between the elements of an Information System with a star network topology.

In the Topology profile, we have nodes (represented by stereotype Node) connected by links that can be local (LocalEdge) if they connect nodes from the same star with its central node, or remote (Edge) if they connect central nodes (MainNode) between each other [2]. Each node is identified by its position (location) and each central node has a state kind (state) that defines their availability (its values are defined by enumeration StateKind). An

end node (EndNode) is also identified by its position (position). The Tree profile has nodes (represented by stereotype Node) connected by links (Edge) to node, end node (Leaf) or root node (root) that has a state kind (state) which defines their availability (its values are defined by enumeration StateKind). Each node is determined by its name (name) and value (value). Moreover, it is possible to perform search operation (Search).

Before merging Tree and Topology, we should necessarily compare the input profiles in order to merge such profiles efficiently. To do this, we need to be able to identify correspondences among UML profile elements in a coherent manner. For example, despite the *Tree.Leaf* and *Topology.EndNode* stereotypes have different names, could they be considered domain concepts of equal semantic values?

B. Contributions of this Paper

To put model comparison in practice involves answering several model comparison questions. As stated in [9], what criteria should we use for identifying correspondences between different models? And how can we quantify these criteria? Considering two input models, should the model comparison techniques produce only one possible result that representing the correspondence among their elements? What properties of the input models should be considered in their match? What should be used so that we can compare models?

The answers for such questions are the contributions of this paper that consist in the definition of a flexible model comparison technique based on *match strategies*. The strategies are implemented by a *match operator* that uses of a range of heuristics including typographic similarities, equivalence among the semantic values of the input model elements and model signature. We propose a brief guidance to specify as conduct the model comparison process. Our approach is constituent of a UML profiles composition mechanism [12] that was shown to be an effective and flexible way for specifying correspondences among UML profiles. Moreover, we specify the approach using the formal specification language Alloy [5] and its tool (the Alloy Analyzer) in order to realize an automatic analysis of the approach.

The remainder of the paper is organized as follows. Section 2 briefly describes the background and the major challenges that researchers face when attempting to realize model comparison. Section 3 presents the our approach based on match strategies and the definition of the match operator. Section 4 presents a brief guidance for model comparison. Section 5 describes the related work. Finally, Section 6 shows some early conclusions and future works.

II. Background and Challenges

Model comparison arises as an essential activity to put the composition in practice and it can be viewed as a generic operation that varies from application to application, in which elements from M_A and M_B are compared in different forms depending of the kind of application. For example, the matching of statechart specifications [9] and of different versions of UML diagrams [11] presents particularity because the artifacts, that are being compared, have different properties, so the model comparison technique is tailored in agreement to them.

The UML specification [14] defines and presents the modelers with the *Profile mechanism* has been specifically specified for providing a lightweight extension mechanism to the UML standard. For instance, we can add semantics that is left unspecified in the metamodel, give a terminology that is adapted to a particular platform or domain and add information that can be used when transforming a model to another model or code.

However, the UML built-in composition mechanism, package merge, is not able to merge profiles or compare the input models correctly. So some research questions arise: how can we compare two profile elements? What activities should we perform to match two input models? Once we have added semantics that does not exist to a UML metamodel element, how can we compare it in a flexible manner?

To the best of our knowledge, the need for comparing models in a flexible manner neither have been pointed out nor even proposed by current model comparison techniques in the model composition mechanisms. This fact shows the pioneer side of this work.

Based on previous works [13], [12] and relevant approach studied (described in Section V), we observed and concluded that the major challenges that researches face when attempting to put into practice the model comparison in the context of MDD can be grouped into the following categories:

- The domain-specific model comparison challenge: Such challenge arises from concerns associated with providing DSMLs for creating and using domain-specific models in the MDD vision. For example, the UML supports two forms of extensions: (1) using profiles to define UML variants and (2) associating particular semantics to specified semantic variation points [14], [4]. Hence, a challenge would be how to develop support for tailoring the model comparison techniques to the semantics plugged into UML semantic variation points and the specializations of the UML metamodel specified by the profiles
- The abstraction level challenge: Once the MDD vision manipulates models in different abstraction levels, how should the model comparison techniques provide support for matching models expressed in different abstraction-level? This challenge poses its problems with respect to understanding and evolving the model comparison techniques across different modeling languages, where each one has its particularity.
- The semantic and properties challenge: As the models have a semantic value associated with it, a pair of them with the same name under matching packages could be assumed to form a match. However, what should be done if they have different semantic values or different properties? For example, two input UML

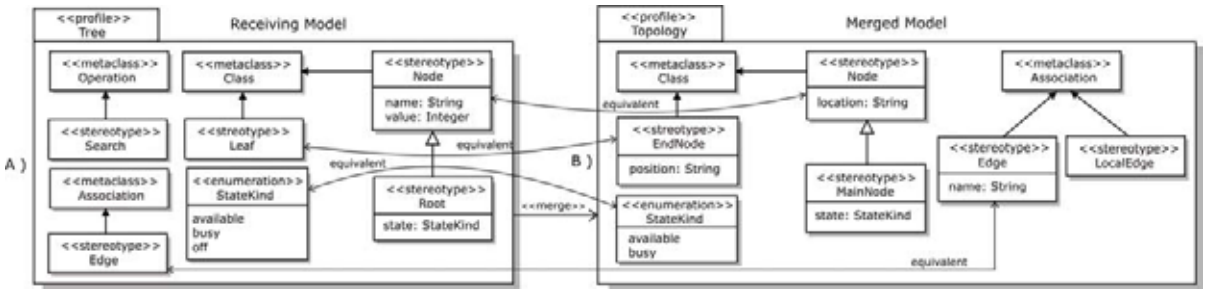


Fig. 1. Example of UML profiles comparison

classes with same name, however one is *abstract* and the other is *concrete*. While the pair of classes may still be considered a match, there is a conformance mismatch between them.

III. Strategy-Based Model Comparison

Having explained a motivation example and defined the challenges of model comparison we present, in this section, a flexible model comparison approach based on *match strategies*. We specified three strategies (i) *default*, (ii) *partial* and (iii) *complete match strategy*; however, new strategies may be created and inserted in our approach as well. We also define a *match operator* that is responsible for putting the strategies in practice together. From input models and the match strategy specification, the match operator verifies the equivalence degree among the input model elements and according to a *threshold* specifies the match models.

A. The Match Operator

The match operator is a heuristic and its goal is to find correspondences among model elements founded in static matching and to implement the *match strategies*. The static matching uses *synonym dictionary*, *model signature* and *typographic similarity* among input model elements in order to define the *equivalence degree* ($\mathcal{S}$).

With a *synonym dictionary* it is possible to make a mapping among the domain concepts that have the same semantic values. The *synonym dictionary* paves the way to the domain specialists to apply their domain expertise in the matching process, once they have defined what concepts are synonyms. Hence, this fact improves the result of the comparison. We denote by $\mathcal{D}(r,m) \rightarrow [0,1]$ the degree of similarity between receiving (r) and merged (m) model elements, it returns 0 whether r and m are synonym, otherwise it returns 1. $\mathcal{D}$ is calculated for every possible pair of (r,m). Initially, every pair (r,m) of input model elements are assumed to be not a synonym, then $\mathcal{D}(r,m) = 0$ for every pair of (r,m). For instance, according to synonym dictionary (see Table I) the stereotypes *Tree.Leaf* and *Topology.EndNode*, depicted in Figure 1(a), represent the same concepts, therefore $\mathcal{D}(Leaf,EndNode) = 1$.

The goal of *typographic similarity* is to determinate $\mathcal{T}(r,m) \rightarrow$ [0..1] to every possible pairs of receiving (r) and merged (m) model elements. The N-gram algorithm [8] is applied to assign a similarity value in [0..1] to every possible pairs of (r,m). These pairs are defined by cartesian product of (R×M), where R and M are the set of receiving and merged model elements, respectively. The result of this is the matrix shown in Figure 2. This algorithm yields a similarity degree to a pair of strings based on counting the number of their identical substrings of length N (we use N = 2).

The *signature* is defined in terms of model element syntactic properties, where a syntactic property of a model element defines its structure. The signature is a collection of values for a subset of syntactic properties in a model element's metamodel class. For example, *isAbstract* is a syntactic property defined in the metamodel class called *Class*. If an instance of a *Class* is an abstract class then *isAbstract* = *true* for the class, otherwise the instance is a concrete class, *isAbstract* = *false*. The set of syntactic properties used to determine a profile element's signature is called *signature type*, as defined in [15]. A signature that consists of all syntactic properties associated with a model element is called *complete signature type*, based on a range of syntactic properties is called *partial signature type* and the signature only based on name is called *default signature type*.

The signature is structured in comparison levels organized hierarchically. For instance, in Figure 1, a possible definition of levels for the stereotype *Tree.Node* would be: *Tree.Node* (name) (level 2), with *Tree.Node.name* and *Tree.Node.value* (tagged values) (level 1). Every profile element type has one signature which is defined for it.

TABLE I
Example of synonym dictionary

Name	Synonym
Leaf	EndNode, FinalNode
Edge	Border, Limit, Margin
Search	Research, Searching, Query

The similarity degree based on signature $\mathcal{M}$ between receiving (r) and merged (m) model element $\mathcal{M}$(r,m) is defined by computing the weighted average between the arithmetic average of the levels (see Equation 1):

$$\mathcal{M} = \frac{\sum_{i=1}^{n} p_i \cdot \left[\sum_{j=1}^{k} \frac{\varphi_{i,j}}{k} \right]}{\sum_{i=1}^{n} p_i} \rightarrow [0..1] \quad (1)$$

- n is the number of levels employed to compare the elements, where $n \geq 1$ and $n \in \mathbb{N}_+^*$.
- p_i represents the weight, being $p_i = i$, where $i \geq 1$ and $i \in \mathbb{N}_+^*$; k expresses the number of elements in each level, where $k \geq 1$ and $k \in \mathbb{N}_+^*$ (i.e. *Tree.Node* has two properties, as these properties represent a level, so $k = 2$);
- $\varphi_{i,j}$ (i and j represent the level and item of model elements that are being compared, respectively) is used to denote if an item of receiving model element (i.e., *name:Strig* in *Tree.Node*) is equivalent to another item of merged model element. It is a boolean variable and we use the match rules (described as follows) in order to assign value to it. The match rules compare items of model elements, so it returns 1 if the rule is satisfied, otherwise it returns 0. For instance, when we compare the *Tree.Root* and *Topology.MainNode* stereotypes, $\varphi_{2,1} = 0$, applying the match rule MR1, and $\varphi_{1,1} = 1$, applying the match rule MR3.

We denote by $\mathcal{S}$ the degree of similarity between receiving (r) and merged (m) model elements. To define the similarity degree it is necessary to combine the partial similarity degrees. To do this, it is calculated the average of $\mathcal{D}$, $\mathcal{T}$, and $\mathcal{M}$, as showed in Equation 2. If $\mathcal{D} = 1$, then $\mathcal{T}$ also assumes value 1 and contrariwise.

$$\mathcal{S} = \frac{(\mathcal{D} + \mathcal{T} + \mathcal{M})}{\mathcal{D} + 2} \rightarrow [0..1] \quad (2)$$

Based on the Equation 2, we compute the similarity degree of every *Tree* elements in related to *Topology* elements. The Figure 2 shows the match results. To produce a correspondence relation between the two models, we set a threshold (t = 0.7). So, pairs of model elements with similarity degree above threshold are considered equivalent. In short, if $\mathcal{S}(r,m) > t$, then r and m are *equivalent*. In Figure 2, we point out the similarity degree above threshold and define the profile elements are equivalent, as follows: (Tree.Node, Topology.Node), (Tree.Edge, Topology.Edge), (Tree.Leaf, Topology.EndNode) and (Tree.StateKind, Topology.StateKind)

		Topology Profile					
		Node	MainNode	Edge	LocalEdge	EndNode	StateKind
Tree Profile	Node	0,83	0,22	0	0,08	0	0
	Root	0	0,05	0	0	0	0
	Edge	0	0	1	0,22	0	0
	Search	0	0	0	0	0	0
	Leaf	0	0	0	0	1	0
	StateKind	0	0	0	0	0	0,86

similarity degree above the threshold ($t = 0.7$)

Fig. 2. Similarity degree between profile elements

B. Match rules

In order to check if two input model element are equivalent, we defined *match rules*. The match operator is responsible to execute these match rules and, according to the resulting of this execution, it defines consequently the value of $\varphi_{i,j}$, which was specified earlier. For every model element and item of model element are necessary a match rule to check if they are equivalent. This checking is based on their signature. If a match rule fails, then the models are not equivalent ($\varphi_{i,j} = 0$). Otherwise, models are equivalent ($\varphi_{i,j} = 1$). The match rules verify whether the input model element properties have the same values, and for each match strategy is defined a set of match rule according to respective signature type of the strategy.

There are three kinds of match rules: (i) *default match rules* are a set of rules that compare models based on only their name, using the default signature type; (ii) *partial match rules* are also a set of rules that compare models based on a number of syntactic properties of the models, using the partial signature type; (iii) *complete match rules* are also a set of rules that compare models based on their syntactic properties, using the complete signature type. Thus, the match operator makes use of these rules to implement the default, partial and complete match strategies. For example, the match operator makes use of the default match strategy (hence using default match rules) to produce the similarity table depicted in Figure 2.

Now, we present a short description of the default match rules used in the motivation example, as follows:

MR1. Stereotype match rule:
MatchStereotype(Stereotype rcv, Stereotype mrgd) →
rcv.name = mrgd.name AND
MatchAttribute(rcv, mrgd) AND
MatchOperation(rcv, mrgd)

MR2. Association match rule:
MatchAssociation(Association rcv, Association mrgd) →
(rcv.name = mrgd.name) AND (rcv.memberEnds =
mrgd.memberEnds)

MR3. Attribute match rule:
MatchAttribute(Stereotype rcv, Stereotype mrgd) →
(rcv.ownedAttribute.name = mrgd.ownedAttribute.name)
AND (rcv.ownedAttribute.TypedElement = mrgd.
ownedAttribute.TypedElement)

MR4. Operation match rule:
MatchOperation(Stereotype rcv, Stereotype mrgd) →
(rcv.
ownedOperation.name = mrgd.ownedOperation.name)
AND (rcv.ownedOperation.ownedParameter.length =
mrgd.ownedOperation.ownedParameter.length) AND
(∀x(rcv.ownedOperation.ownedParameter[x] =
mrgd.ownedOperation.ownedParameter[x])

MR5. Enumeration match rule:
MatchEnumeration(Enumeration rcv,
Enumeration mrgd) → rcv.name = mrgd.name AND
MatchEnumerationLiteral(Enumeration rcv,
Enumeration mrgd)

MR6. Enumeration Literal match rule:
MatchEnumerationLiteral(Enumeration rcv, Enumeration mrgd) → ∀x(rcv.ownedLiteral.name[x] = mrgd.ownedOperation.name[x])

IV. A Guidance for Model Comparison

There is little agreement on requirements, activities and steps that should be followed in order to accomplish the model comparison, and even less on good practices to avoid errors during matching. Several works (e.g., see [7], [11]) have been proposed to tackle the problems found in model comparison, but none of them, as yet, was defined as standard. In [14], the UML built-in model comparison technique does not present a task flow to help the comparison specification of UML models, does not present a good documentation, and does not define how model comparison should be performed.

We previously identified and delegated activities to the match operator. We aim to successfully order and provide a flow of how such activities are accomplished. Such flow can be used as a *guidance* to compare models, and it aims to represent good practices and become as comprehensive as possible the match operator role in the model comparison process.

The guidance is organized in two phases: (1) *initial* and (2) *comparison phase*. The *initial phase* is started up when the matching operator receives the input models. The match operator analyzes the models in order to know each type (i.e. Stereotype, Class, Association, etc). Such models are separated and grouped according to their types. For example, Stereotypes (Tree.Node and Topology.Node) and Association (Tree.Edge and Topology.Edge) are identified and grouped according to their types.

The goal of the *comparison phase* is to define what input model elements are equivalent. It is initially realized as an analysis of the input models and a signature is defined for every model element type. The next step is to specify the match strategy that determines how the comparison will be accomplished. The match operator defines the similarity degree ($\mathcal{S}$) for every receiving and merged model element, and based on a threshold (t) finally it determines model elements are equivalent. The phase is finished as soon as the matching models, no-matching models and matching description are specified. The next step is to merge the models, however this activity is not the focus of this paper.

V. Related Work

The model comparison is applied in different domains and contexts, and plays a central role in numerous applications, such as model composition, schema integration, schema evolution and migration, merging of source code, application evolution, database integration, differences between XML documents, and differences between versions of UML diagrams. Thus, previous research works have proposed many techniques to tackle the inherent problems related to matching, and achieved an automation degree of the match operation for specific application domains. We

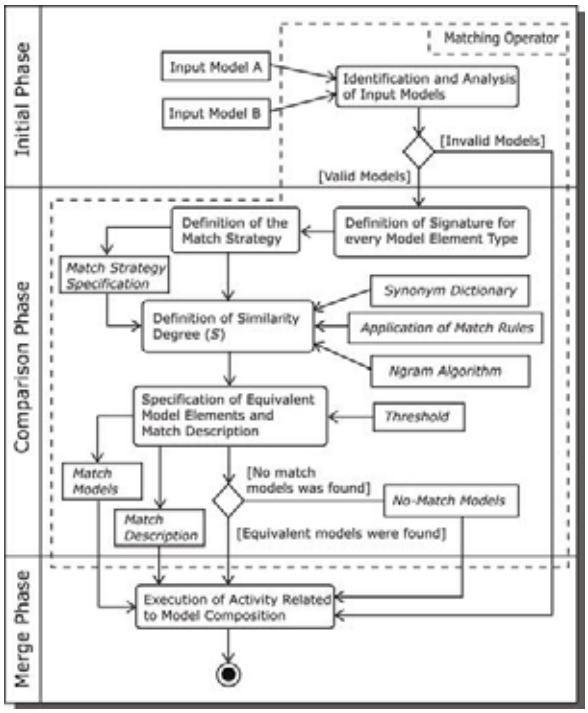


Fig. 3. A guidance for model comparison

give an overview on other relevant approaches related to our goals of putting flexibility into the model comparison process and analyze others that make use of model comparison to merge models. To do this, the main focus of each approach is summarized briefly, followed by pointing out similarities and differences to our own approach (see Figure 4).

Model Composition Semantics. S. Clarke [1] introduces composition semantics for UML class diagrams. The approach defines a new design construct, called *composition relationship* that supports the specification of how design models should be composed. With this *composition relationship* it is possible to: (*i*) identify and specify overlapping and non-overlapping concepts; (*ii*) specify how models should be integrated, and how conflicts in equivalent elements are reconciled. The identification of the overlapping parts is based on the name of the input models; it is a weakness of the approach.

Model Composition Directives. Reddy et al. [15] present a model composition technique relies on signature matching, in which model elements are merged if their signatures are correspondent. However, the match operator, in our work, makes use of a static matching approach based on synonym dictionary, typographic similarity and model signature in order to define the degree of similarity between two models elements.

Package Merge. It is the composition mechanism of the UML [14] and is defined by *match rules*, *constraints* and *transformation* (the merge rules). The major application is in the implementation of the UML compliance levels. In principle, their *match rules* are similar to *match* used by our *match operator*. However, its *match rules*

are expressed in natural language and the match process consider only the name of the models. Moreover, the definition of Package Merge is incomplete, ambiguous and inconsistent.

Epsilon Merging Language. EML [6] is a metamodel agnostic language for expressing model composition. It includes a model comparison and model transformation language as subsets. The model comparison is only based on syntactic criterion. However, the match, in our approach, is founded in synonym dictionary, typographic similarity, syntactic properties and match strategy.

Difference between Models. It presents an approach of the how to detect and visualize differences between versions of UML documents such as class or object diagrams. It produces a unified document which contains the common and specific parts of two base documents, where the specific parts are highlighted [11]. While our approach tackles a range of very difficult problems related to dealing with comparison of semantics values in a flexible manner, it is primarily concerned with the comparison and manipulation of models from the same domain and with equal semantic values; without any flexibility during the comparison.

Approaches	Assessment Criteria: Guidance	Strategies	Typography	Name	Structural	Semantic	Rules	Operator
Model Composition Semantics				X				
Model Composition Directives				X	X			
Package Merge				X			X	
Epsilon Merging Language					X			
Difference between Models				X	X			

Legend:
X – supported by the approach
☐ – not supported by the approach

Fig. 4. Comparison of related approaches

VI. Conclusions and Future Work

In this paper we discussed the importance of model comparison for the task of model composition, its problems and challenges involved in its implementation. Our approach provides a flexible form of realizing the model comparison founded on match strategies by defining the match operator and by specifying its responsibility. Moreover, we consider that the range of different forms for matching models improves and assures a better performance to the comparison process and the use of guidance in order to provide a clear and easy manner to perform the comparison helps its improvement and evolution.

The problems and challenges outlined throughout the paper should encourage researchers to cope with the ever-present problem of matching models so that new generation of the application can enjoy the use of better techniques. Our approach has some limitations that should be investigated further. When models are defined, it is possible to associate them semantics constraints. These constraints should be considered and respected when it is necessary to perform the composition so that the specified semantic is not disrespected. Thus, our approach is not able, as yet, to compare these constraints. We claim to enhance the functionality of the match operator by creating new match strategies and improving the match rules. Another extension of our approach would be the use of ontology to improve the handle of the models' semantic values.

Even through our approach has been implemented and integrated to a profile composition mechanism demonstrating feasibility [12], empirical studies are necessary to validate the approach in real world design settings of model comparison and verify its performance and applicability in different application domains. Finally, we observed improvement in model comparison is absolutely necessary to the model engineering evolution and to allow model engineering to become an industrial reality.

References

[1] S. Clarke, "Composition of Object-Oriented Software Design Models," Ph.D. dissertation, School of Computer Applications, Dublin City University, Dublin, Irland, January 2001.

[2] L. Fernndez and A. Moreno, "An Introduction to UML Profiles," in *The European Journal for the Informatics Professional*, vol. 5, no. 2, April 2004, pp. 6–13.

[3] R. France, S. Ghosh, and T. Dinh Trong, "Model Driven Development Using UML 2.0: Promises and Pitfalls," *IEEE Computer Society*, vol. 39, no. 2, pp. 59–66, February 2006.

[4] R. France and B. Rumpe, "Model-Driven Development of Complex Software: A Research Roadmap," in *Future of Software Engineering (FOSE'07) co-located with ICSE'07*, Minnesota, EUA, May 2007, pp. 37–54.

[5] D. Jackson, "Alloy: a Lightweight Object Modelling Notation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 11, no. 2, pp. 256–290, 2002.

[6] D. Kolovos, "Epsilon Merging Language Project Page," http://www.eclipse.org/gmt/epsilon/.

[7] D. Kolovos, R. Paige, and F. Polack, "Model Comparison: a Foundation for Model Composition and Model Transformation Testing," in *International Workshop on Global Integrated Model Management*. New York, NY, USA: ACM Press, 2006, pp. 13–20.

[8] C. Manning and H. Shütze, *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[9] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave, "Matching and Merging of Statecharts Specifications," in *ICSE'07*, Minnesota, EUA, May 2007, pp. 54–64.

[10] Object Management Group, *MDA Guide Version 1.0.1*, 2003, http://www.omg.org/docs/omg/ 03-06-01.pdf.

[11] D. Ohst, M. Welle, and U. Kelter, "Differences between Versions of UML Diagrams," in *9th European Software Engineering Conference*. ACM Press, 2003, pp. 227–236.

[12] K. Oliveira, "Composition of UML Profiles," Master's thesis, Informatics Faculty, Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil, February 2008.

[13] K. Oliveira and T. Oliveira, "A Guidance for Model Composition," in *International Conference on Software Engineering Advances (ICSEA'07)*, 2007, pp. 27–32, IEEE Computer Society.

[14] OMG, *Unified Modeling Language: Infrastructure version 2.1*, Object Management Group, February 2007.

[15] Y. Reddy, R. France, G. Straw, N. M. J. Bieman, E. Song, and G. Georg, "Directives for Composing Aspect-Oriented Design Class Models," *Transactions of Aspect-Oriented Software Development*, vol. 1, no. 1, pp. 75–105, 2006.

[16] S. Sendall and W. Kozaczynski, "Model Transformation: The Heart and Soul of Model-Driven Software Development," *IEEE Software*, vol. 20, no. 5, pp. 42–45, 2003.

Towards Metrics for Ontology Balance

Steffen Mencke, Martin Kunz, and Reiner R. Dumke

Abstract— **Nowadays, measurement and assessment of artifacts within the area of software development are of high concern for industrial organizations as well as for scientific institutions. Ontologies are a fundamental concept of the Semantic Web as envisioned by Tim Berners-Lee. Together with an explicit representation of the semantics of data for machine-accessibility such domain theories are the basis for intelligent next generation applications for the web and other areas of interest.**

The balance of ontology's is on higher interest because the usability and convertibility of ontologies is strongly related to the manner how the elements are arranged. This paper takes into account existing metrics and tries to present new ideas, as well.

At first this paper contains a brief description and categorization of existing ontology metrics with a focus on applicability regarding the balance of ontology's taking into account structure and knowledge related aspects. Therefore a Goal-Question-Metric-based procedure was used. In a second step initial ideas for additional metrics are identified and enriched with certain presented formulas. A third step expresses different approaches for further research work: gravity-related and weighted-graphs-based approaches towards metrics for ontology balance. The paper's conclusion presents certain use cases for the application of balanced ontologies in the area of e-learning systems.

Index Terms—**Balance, Classification, Metric, Ontology**

I. Introduction

THE importance of measuring artifacts emerging during the software development process is beyond controversy not only for economic purposes.

Ontologies are a fundamental concept of the Semantic Web envisioned by Tim Berners-Lee [1]. Together with explicit representation of the semantics of data for machine-accessibility, such domain theories are the basis for intelligent next generation applications for the web and other areas of interest [2] with a special focus on knowledge sharing and reuse. Ontologies are also basis for interaction and work of different agents or applications [3]. Top-level application areas identified by [4] are collaboration, interoperation, education and modeling.

Ontologies can be defined as a specification of a conceptualization [5], or in other words as the formal representation of an abstract view of the world. They include a vocabulary, instances, taxonomy, relations and axioms about a certain domain.

A vocabulary defines terms with unambiguous meanings. Furthermore, logical statements for the description of terms and rules for their combination and relation are provided. A taxonomy is part of the ontology concept for a hierarchical classification in a machine-processable form. Individuals/instances represent the objects of the ontology and thereby the available knowledge, while classes/concepts describe abstract sets of individuals. Attributes can be assigned to instances for description. They have a name and value. The last key concept of ontologies is the relation. It can be described by using attributes and assigning another individual as a value. Common relation types are the is-a relation (subsumption relation) and the part-of relation (meronymy relation). The possibility to define special domain specific relations is a considerable additional value of the concept of an ontology. Axioms are always true and represent knowledge that is not inferable from other individuals.

It is possible to distinguish ontologies in two broad categories: lightweight and heavyweight ontologies. A lightweight ontology is described by individuals, classes, attributes, relations and axioms, meanwhile heavyweight ontologies are an extension of lightweight ones by the additional usage of axioms for a more detailed domain description.

There already exist many ontologies. Some are available via libraries like the DAML ontology library [6] and the SchemaWeb library [7].

After this short introduction in the field of ontologies, the authors analyze the structure of ontologies to map existing software metrics for their applicability in this field of research in section II following a GQM approach. Furthermore, existing metrics are classified. Section III dedicated to a special field of ontology metrics which the authors found rarely researched so far – the balance of ontologies. This paper ends with some conclusions and remarks about future work in section 4.

II. Classification of Existing Ontology Metrics

For the purpose of measuring the Goal Question Metric (GQM) approach [8] helps in discovering adequate

Manuscript received March 1, 2008.

S. Mencke is with the Otto-von-Guericke University of Magdeburg, 39106 Magdeburg, Germany (corresponding author to provide phone: +49-(0)391-67-12705; fax: +49-(0)391-67-12810; e-mail: mencke@ ivs.cs.uni-magdeburg.de).

M. Kunz is with the Otto-von-Guericke University of Magdeburg, 39106 Magdeburg, Germany (e-mail: makunz@ ivs.cs.uni-magdeburg.de).

R. R. Dumke is with the Otto-von-Guericke University of Magdeburg, 39106 Magdeburg, Germany (e-mail: dumke@ ivs.cs.uni-magdeburg.de).

measurement attempts and goals. Initially, it requires the definition of precise goals to form the foundation for the nomination of questions suitable for discussing issues from different viewpoints. Finally, metrics qualified for answering these questions become apparent. Afterwards a tailored measurement as well as its evaluation concerning goal attainment is possible.

The quantification of metrics attributes is separated into two different areas being divided into four major scopes. These areas are scheme-related and content-related, respectively.

At first it is analyzed which metrics are used to measure the content of ontologies. One can identify two major goals in this area: the granularity of the enclosed content and the coverage of the content (see figure 1).

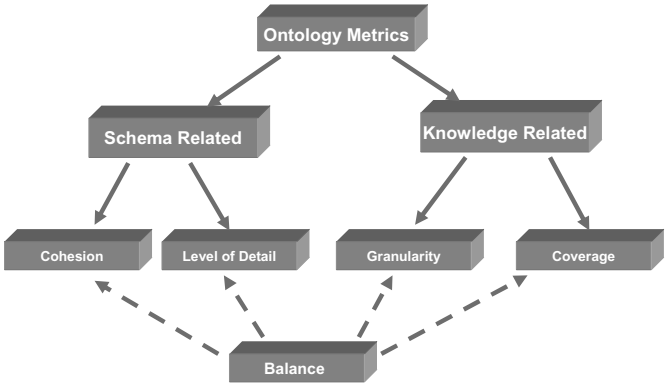


Fig. 1. Genealogy of ontology metrics

To achieve these goals the mentioned GQM approach is used to identified the content granularity and content coverage metrics as shown in table I and table II. In the second area (the structures of ontologies) two goals were identified as well.

An aspect which is well described by existing metrics is the structure of ontology and identified major scopes are the level of detail and cohesion. Especially a scheme-based level of detail is important to evaluate ontology because it is fundamental to achieve content granularity (see table III). Having introduced this concept as an indicator for information distribution, another one is needed to describe coherence of distinct classes. It quantifies relation-based information in ontology. Chosen metrics are presented in table IV.

It is possible to evaluate the structure of ontologies taking into account these two goals. Other approaches like scheme completeness and scheme granularity are not useful because of different reasons. So scheme completeness, when creating a completely new ontology, is a semantic question which can not be answered by using metrics. One can target this question by empirical analyses in ontology usage by taking into account other domain related ontologies. The question whether an ontology is complete or not can not be finally answered by using the ontology itself. The analysis in this direction depends very much on a subjective point of view.

TABLE I
CHOSEN CONTENT GRANULARITY RELATED METRICS

Name of Metric	Formula	Description
Average Population (Pop) [9]	$Pop = \|I\| / \|C\|$, with $\|I\|$ as the number of instances in the knowledge base and $\|C\|$ as the number of classes defined in the ontology.	This metric may serve as an indication of the number of instances compared to the number of classes.
Cohesion (COH) [9]	$COH = \|SCC\|$ as the number of separate connected components	This indicates what areas need more instances in order to enable instances to be more closely connected.
Connectivity (Cn) [9]	$Cn = \|I_j, P(I_i, I_j) \wedge I_j \in C_i(I)\|$ as the number of instances of other classes that are connected to instances of that class (I_j).	It is an indication of the number of relationships instances of each class to other instances.

TABLE II
CHOSEN CONTENT COVERAGE RELATED METRICS

Name of Metric	Formula	Description
Class Richness (CR) [9]	$CR = \|C'\| / \|C\|$, with $\|C'\|$ as the number of classes used in the base and $\|C\|$ as the number of classes defined in the ontology.	Describes how instances are distributed across classes.
Density measure (DEM) [10]	$DEM = \frac{1}{n}\sum_{i=1}^{n} w_i\|C_{Sub}\| + w_i\|C_{Sup}\| + w_i\|C_S\| + w_i\|I\| + w_i\|P\|$, with C_{Sub} as the number of a class' subclasses, C_{Sup} as the number of its superclasses, C_S as the number of its siblings, I as the number of its instances, P as the number of its relations, and w_i as a weight factor.	This metric indicates how well a given concept is defined in the ontology.
Relation-ship Richness (RRC) [9]	$RR_C = \|P(I_i, I_j), I_i \in C_i(I)\| / \|P(C_i, C_j)\|$, with $(P(I_i, I_j))$ as the number of relationships that are being used by instances I_i that belong to C_i, and $(P(C_i, C_j))$ as the number of relationships that are defined for C_i at the schema level.	Identifies how well the extraction process performed in the utilization of information is defined at the schema level.
Impor-tance (IMP) [9]	$IMP = \|C_i(I)\| / \|I\|$, with $\|C_i(I)\|$ as the number of instances that belong to the subtree rooted at C_i in the knowledge base, and $\|I\|$ as the number of instances in the knowledge base.	It is not an exact measure, but it can give a clear idea on what parts of the ontology are considered focal and what parts are on the edges.
Fullness (F) [9]	$F = \|C_i(I)\| / C_i'(I)$, with $\|C_i(I)\|$ as the actual number of instances that belong to the subtree rooted at C_i, and $C_i'(I)$ as the expected number of instances that belong to the subtree rooted at C_i.	Describes how well was the data extracted with respect to the expected number of instances of each class.

TABLE III
CHOSEN SCHEME-BASED LEVEL OF DETAIL RELATED METRICS

Name of Metric	Formula	Description
Attribute Richness (AR) [9][11]	$AR = \lvert A \rvert / \lvert C \rvert$, with $\lvert A \rvert$ as the number of attributes of all classes and $\lvert C \rvert$ as the number of classes.	This metric can indicate the quality of ontology design.
Centrality Measure (CEM) [10]	$CEM = \frac{1}{n}\sum_{i=1}^{n} 1 - \left\lvert \frac{D[C] - \frac{H[C]}{2}}{\frac{H[C]}{2}} \right\rvert$ with $H[C]$ as the longest path that contains the class C from root of the branch to its bottom node, and $D[C]$ as the length of the path to C from the root.	For this metric it is assumed that mid-leveled classes tend to be more representative for an ontology due to more details and prototypical character.
Number of Leaf Nodes (NoL) [12]	$NoL = \lvert C_j \rvert$, with $1 \le j \le n$ and C_j leaf class of the ontology.	A leaf class has no semantic subclass explicitly defined in the ontology.
Number of Root Classes (NoR) [12]	$NoR = \lvert C_j \rvert$, with $1 \le j \le n$ and C_j root class of the ontology.	A root class in an ontology means the class has no semantic super class explicitly defined in the ontology.
Average Depth of Inheritance Tree of Leaf Nodes (ADIT-LN) [12]	$ADIT - LN = \lvert D_j / n \rvert$, with $1 \le j \le n$ and D_j as total number of nodes on j^{th} path.	This metric describes the sum of depths of all paths divided by the total number of paths.

TABLE IV
CHOSEN SCHEME COHESION RELATED METRICS

Name of Metric	Formula	Description
Relationship Strength (RSS^O) [13]	$RSS^O(P,Q) = \max_{u \in cl(P), v \in cl(Q)} \{RSS(u,v)\}$ with: P and Q as the classes of interest and $cl(P)$, $cl(Q)$ as the sets of all concepts assigned to the classes P and Q, and $RSS(C_1, C_2) = \frac{\text{maxDepth}}{\text{maxDepth} + \gamma} * \frac{\alpha}{\alpha + \beta}$.	Describes strength of relationship between two classes.
Relationship Richness (RR) [9]	$RR = \lvert P \rvert / \lvert SC \rvert + \lvert P \rvert$, with P as the number of relationships, and SC as the number of sub-classes (= inheritance relationships).	Describes the diversity of relations and placement of relations in the ontology.
Inheritance Richness (IR_C) [9]	$IR_C = \frac{\sum_{C_i \in C} \lvert H^C(C_1, C_i) \rvert}{\lvert C' \rvert}$, with $\lvert H^C(C_1, C_i) \rvert$ as the number of subclasses (C_1) of a class C_i, and $\lvert C' \rvert$ as the number of nodes in the subtree.	Describes the distribution of information in the current class sub-tree per class.

III. METRICS FOR THE BALANCE OF ONTOLOGIES

Having presented four starting points for the evaluation of ontologies in the following another general aspect concerning the structure and the content of ontologies is introduced: the balance of a distinct ontology (cp. figure 1). Existing measures in this area (for example Average Depth, Average Breadth) can not completely quantify ontology aspects concerning the balance. The balance of ontology is important because it is to be used as an indicator how good the ontology is built up and one can identify anomalies by analyzing the balance.

However research efforts in this area are very rare and a complete framework for balancing ontologies is missing. In the following initial instruments for quantifying ontologies concerning the balance are presented.

Concerning the balance of ontologies there exist different general aspects that can be helpful to quantify an ontology's balance.

- Classes:
  - Equal number of subclass in equal level of abstraction
    $\lvert C_{Sub} \rvert_L^{C_i} \approx \lvert C_{Sub} \rvert_L^{C_j}$ with $i, j = 1,...,n \wedge i \neq j$
  - Equal number of subclass in different subtrees
    $\left\lvert C_{Sub_k} \right\rvert^{C_i} \approx \left\lvert C_{Sub_l} \right\rvert^{C_i}$ with $i,k,l = 1,...,n$
- Relations:
  - Equal number of relations in equal level of abstraction
    $\lvert P \rvert_L^{C_i} \approx \lvert P \rvert_L^{C_j}$ with $i, j = 1,...,n \wedge i \neq j$
  - Equal number of relations in different subtrees
    $\left\lvert P_{C_{Sub_k}} \right\rvert^{C_i} \approx \left\lvert P_{C_{Sub_l}} \right\rvert^{C_i}$ with $i,k,l = 1,...,n$
- Attributes:
  - Equal number of attributes in different concepts in equal level of abstraction
    $\lvert A \rvert_L^{C_i} \approx \lvert A \rvert_L^{C_j}$ with $i, j = 1,...,n \wedge i \neq j$
  - Equal number of attributes in different subtrees
    $\left\lvert A_{C_{Sub_k}} \right\rvert^{C_i} \approx \left\lvert A_{C_{Sub_l}} \right\rvert^{C_i}$ with $i,k,l = 1,...,n$
- Instances:
  - Equal number of instances of different concepts in equal level of abstraction
    $\lvert I \rvert_L^{C_i} \approx \lvert I \rvert_L^{C_j}$ with $i, j = 1,...,n \wedge i \neq j$
  - Equal number of instances in different subtrees
    $\left\lvert I_{C_{Sub_k}} \right\rvert^{C_i} \approx \left\lvert I_{C_{Sub_l}} \right\rvert^{C_i}$ with $i,k,l = 1,...,n$
- Subtrees:
  - Equal depth of each subtree
    $\left\lvert DIT_{C_{Sub_k}} \right\rvert^{C_i} \approx \left\lvert DIT_{C_{Sub_l}} \right\rvert^{C_i}$ with $i,k,l = 1,...,n$

Besides these tree-based approaches a second set of formulas is presented in the following to analyze balance aspects of ontologies. For this purpose previous published work about the specification of distance-based semantic windows is used [14].

An ontology is defined as $O = (C, R, D, I)$, where C is the set of ontological concepts following a taxonomic structure, $R = R_{tax} \cup R_{ntax}$ is the set of object properties/relations taxonomically and non-taxonomically relating two concepts $R_{ij}(C_i, C_j)$ and D is the set of datatype properties/attributes of the ontology. I is the set of instances. An ontological component of each of these types can be the enrichment point for the semantic window. From this four different aspects the dimensions of the semantic window can be derived.

- Concept view
- Datatype property view
- Object property view
- Instance view

For each of the four views, distance measures are defined for the existing dimensions. A help function is $f^{niv}(C_i)$ describing the level of the concept according to its taxonomic level with $f^{niv}(C_{root}) = 0$. Function $f^{parent}(C_i, C_j)$ delivers back the first more abstract concept shared by C_i and C_j, if it exists and is connected to them only via $R \in R_{tax}$. $f^{tax}(C_i, C_j)$ and $f^{ntax}(C_i, C_j)$ determine the length of the taxonomic or non-taxonomic path of concepts from C_i to C_j (the result is -1, if there does not exist such a path).

The dimensions of the distance related to the ontology's concepts having a concept as the focusing point are defined in (1) to (4). The single distance measures relate to the abstraction dimension distance c^{abs}, to the specialization dimension distance c^{spec}, to the sibling dimension distance c^{sib} and to the non-taxonomic dimension distance c^{ntax}. They measure the distance between the focusing point concept C_F and another concept C_j of the ontology.

$$c^{abs}(C_F, C_j) = f^{niv}(C_F) - f^{niv}(C_j) \quad (1)$$

$$c^{spec}(C_F, C_j) = f^{niv}(C_j) - f^{niv}(C_F) \quad (2)$$

$$c^{sib}(C_F, C_j) = f^{niv}(C_F) - f^{niv}(f^{parent}(C_F, C_j)) \quad (3)$$

$$c^{ntax}(C_F, C_j) = f^{ntax}(C_F, C_j) \quad (4)$$

The equations above are restricted by: $C_F, C_i, C_j \in C$. Equation (1) is restricted by: $f^{niv}(C_F) > f^{niv}(C_j)$ and $f^{ntax}(C_F, C_j) \neq -1$. Equation (2) is restricted by: $f^{niv}(C_F) < f^{niv}(C_j)$ and $f^{ntax}(C_F, C_j) \neq -1$. Equation (3) is restricted by: $f^{niv}(C_F) = f^{niv}(C_j)$ and $f^{niv}(f^{parent}(C_F, C_j)) < f^{niv}(C_F)$.

With this set of described formulas we are able to define first knock-out criterions for balanced ontologies:

- An ontology which contains not a single pair of leaf nodes with no sibling distance can not be balanced
- If every subtree of the root node has a different maximal abstraction dimension the root can not be balanced
- Two concepts having a sibling distance must have the same specialization distance to their leafs

The presented approach is a first analysis of the targeted problem of missing balance metrics for ontologies. The mentioned numerous aspects need to be integrated in a set of formulas. Due to manifold characteristics of the described starting points, one has to do fundamental research about the mathematical base to map the existing complexity of the problem to certain metrics formulas. Knock-out criteria can be a first starting point but it is not sufficient and quality models with distinct measures are desirable.

Related research should follow e.g. the following ideas:

- Gravity-related approach:
  - Identification of a center of gravity
  - Measuring C^{abs}, C^{spec}, C^{sib} and C^{ntax} to the border concepts of the ontology (roots, leafs, …)
  - Ontology is balanced, if $c^{abs} \approx c^{spec} \approx c^{sib} \approx c^{ntax}$
  - Extension towards multiple centers of gravity
- Weighted graphs approach:
  - Determine a weight W_{C_i} for every node of the ontology's graph representation based on instances' size, instances' number, concept's relations and attributes, etc.
  - Ontology is balanced if (a) every node C_i has a similar weight or (b) all nodes on the same abstraction level have a similar weight.

IV. Summary, Conclusion and Future Work

In this paper an overview of existing metric ontology measurement following a structured approached based on the concept of GQM was presented. During research a lack of metrics for balance-measuring for ontologies was observed. To close this gap, different criteria for a balance measuring framework were identified and future steps towards a balance-metrics set were outlined.

A. Conclusions

Measuring just because it is possible can not be an intention. The following ideas present some initial ideas for ontology metrics in certain applications.

The area of knowledge discovery can be a major building-block for e-learning. The creation of courses or the measurement of learning efforts can be revised with ontologies.

Measurement approaches in this direction can be for example:

(L1) The determination of the semantic similarity between an ontology describing the domain to be learned and an ontology created by the learner(s) during the learning process is an approach ***to measure the standard of knowledge*** at a discrete point in time. By repetition the learning progress of the person/community that built up the second ontology can be analyzed for multipurpose reasons.

(L2) Measuring the complexity of evolving ontologies during a learning effort or an examination can help to ***identify concepts that were learned very well*** or were not yet learned.

(L3) The creation of tests and exercises based on ontologies will lead to automatic ***determination of the level of difficulty***, respectively of the complexity of the question and the expected answer based on the ontology complexity.

(L4) Identifying matching concepts in ontologies to ***automatically generate courses described by ontologies is*** another option.

(L5) Another usage for a similarity measure can be the ***description of course content*** depending on a domain ontology.

Agent technology is another very interesting application area. The authors expect ontology metrics to be extremely useful for several aspects, e.g.:

(A1) An ***agent's functionality*** can be characterized by analyzing the used communication ontology.

(A2) It becomes also possible to identify a ***useful separation of functionalities and evolving communication based on an ontology*** containing a service description. Such an approach is useful to automatically identify the mapping of functionalities to agents as postulated in [15] and [16].

(A3) The ***balancing of workload*** becomes possible when the work is effort-driven distributed based on an ontology.

Another mentionable aspect is the usage of appropriate metrics in measurement infrastructures.

(I1) ***Implementation of measurement services*** for the presented metrics to integrate ontology measurement into our service oriented measurement infrastructure is interesting as previously presented in [17] and [18].

B. Future work

There are many open questions regarding ontology metrics as for example maturity (how ready is it to use?), robustness (how it can handle unexpected concepts), language flexibility (how stable is language?) and domain friendliness (how easy is to develop domain ontologies based on an upper ontology?) [19]. In the future the authors will focus on the development of the sketched balance metrics and their application to certain areas.

REFERENCES

[1] T.B. Lee, J. Hendler, and O. Lassila, "The Semantic Web", Scientific American, 284, pp. 34-44, 2001.

[2] V. Devedzic, "Semantik Web and Education", Springer, 2006.

[3] S. Kernchen, D. Rud, F. Zbrog, and R. Dumke, "Processing Remote Measurement Databases by the Means of Mobile Agents", In Proceedings of the 3rd International Conference on Web Information Systems and Technologies, Barcelona, Spain, March 2007.

[4] R. Fikes, "Multi-Use Ontologies", Stanford University (February 07, 2007), http://www.ksl.stanford.edu/people/fikes/cs222/1998/Ontologies/sld001.htm.

[5] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications", Knowledge Acquisition, 5, pp. 199-220, 1993.

[6] DAML, "DAML Ontology Library", http://www.daml.org/ontologies/.

[7] SchemaWeb, "SchemaWeb", http://www.schemaweb.info/.

[8] V. Basili, and D. Weiss, "A Methodology for Collecting Valid Software Engineering Data", IEEE Transaction on Software Engineering, 10, pp. 728-738, 1984.

[9] S. Tartir, I.B. Arpinar, M. Moore, A.P. Sheth, and B.A. Meza, "OntoQA: Metric-Based Ontology Quality Analysis", In Proceedings of IEEE ICDM 2005 Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources, 2005.

[10] H. Alani, and C. Brewster, "Ontology Ranking Based on the Analysis of Concept Structures", Proceedings of the 3rd International Conference on Knowledge Capture, ACM Press, 51-58, 2005.

[11] P. Buitelaar, T. Eigner, and T. Declerck, "OntoSelect: A Dynamic Ontology Library with Support for Ontology Selection", Proceedings of the Demo Session at the International Semantic Web Conference, 2004.

[12] H. Yao, A.M. Orme, and L. Etzkorn, "Cohesion Metrics for Ontology Design and Application", Journal of Computer Science, 1, pp. 107-113, 2005.

[13] X. Wu, L. Zhu, J. Guo, D. Zhang, and K. Lin, "Prediction of Yeast Protein-Protein Interaction Network: Insights from the Gene Ontology and Annotations", Nucleic Acids Research, 34, 2137-2150, 2006.

[14] S. Mencke, Andreas Schmietendorf and R. Dumke, "Distance-Based Semantic Windows", submitted to 6th e-Learning Fachtagung Informatik der Gesellschaft für Informatik (DeLFI 2008), Lübeck, Germany, September 7-10, 2008.

[15] S. Kernchen, F. Zbrog, and R. Dumke, "ABEL-GUI: An Agent-Based Graphical User Interface for E-Learning", In Proceedings of the 3rd International Conference on Web Information Systems and Technologies, Barcelona, Spain, March 2007.

[16] S. Kernchen, and R. Dumke, "Developing Adaptive and Self-Managed Graphical User Interfaces", In Proceedings of the Second International Conference on Interactive Mobile and Computer Aided Learning, Amman, Jordan, April 2007.

[17] M. Kunz, A. Schmietendorf, R. Dumke, and C. Wille, "Towards a Service-Oriented Measurement Infrastructure". Proceedings of the 3rd Software Measurement European Forum (Smef 2006), May, 10-12, Rome, Italy, pp. 197-207, 2006.

[18] M. Kunz, A. Schmietendorf, R. Braungarten, and R. Dumke, "Service-Oriented Adjustment of Test and Measurement Tools" (In german). In: A. Schmietendorf and R. Dumke: Proceedings of 1. Workshop Bewertungsaspekte serviceorientierter Architekturen (BSOA06), 24. November, FHW Berlin, private publishing venture: Otto-von-Guericke-University of Magdeburg, pp. 11-20, 2006.

[19] A. Terry, "SUO: Thoughts on the Ontology Evaluation Questions", http://grouper.ieee.org/groups/suo/email/msg12408.html.

Techniques for De-fragmenting Mobile Applications: A Taxonomy

Damith C. Rajapakse
School of Computing, National University of Singapore
damith@comp.nus.edu.sg

Abstract

Fragmentation, in the context of mobile applications, is the inability to "write once and run anywhere". Fragmentation increases the effort required in all aspects of application development. This paper analyzes various aspects of fragmentation, and presents a taxonomy of techniques used to combat it. Our aim is to establish a set of useful terminology for the benefit of researchers and practitioners working in this area.

1. Introduction

Fragmentation is the term used in the industry to describe the inability to "write once and run anywhere", often resulting in multiple versions of an application. More formally, we define fragmentation as the "inability to develop an application against a reference operating context and to achieve the intended behavior in all operating contexts suitable for the application". Further, we define the *operating context* (OC) for an application as the "external environment that influences its operation". Therefore, an OC is defined by the hardware/software environment in the device, the user, and the environmental constraints introduced by various stakeholders such as the network operator. While fragmentation can affect any type of application, this paper focuses on the fragmentation of mobile applications. Note that by "mobile applications" we mean *installed* applications on the mobile device and not the server-side applications such as SMS-based applications[1] or mobile web applications[2].

Fragmentation is caused by the diversity of OCs (see Figure 1 for an illustration). In Section 2, we describe how one OC could differ from another, resulting in fragmentation. While users, developers, distributors, carriers and device manufacturers are all affected by fragmentation, this paper looks at fragmentation from the point of view of an organization developing mobile applications. In section 3, we describe how fragmentation affects various aspects of mobile application development. As fragmentation is a big problem in the industry today, a number of techniques have emerged to combat it. We call them *de-fragmentation* techniques. Section 4 presents a taxonomy of existing de-fragmentation techniques, based on the basic approach each one uses to tackle the problem. This taxonomy was inspired by the work of practitioners [3] and later refined based on further feedback from practitioners (as acknowledged in Section 7). Where appropriate, we refer to industry tools to illustrate each approach. Comments about related work, conclusions, and future directions are given at the end of the paper.

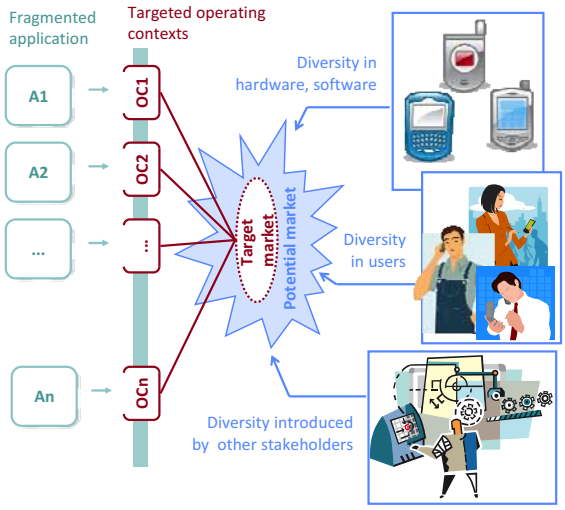


Figure 1. Fragmentation overview

2. Causes of fragmentation

By definition, fragmentation is caused by the diversity of operating contexts (OCs). One operating context may differ from another for the following reasons:

- **Hardware diversity** of the device, such as differences in screen parameters (size, color depth, orientation, aspect ratio), memory size, processing power, input modes (keyboard, touch screen, etc.), additional hardware (camera, voice recorder etc.), and connectivity options (bluetooth, IR, GPRS, etc.).

[1] A server-side application accessed by a mobile device, using SMS as the mode of communication

[2] An application accessed over the Internet, using a web browser on a mobile device.

- **Software diversity**, which may be a result of platform diversity or implementation diversity:
  - **Platform diversity** is caused by factors such as differences in platforms/OS (Symbian, Nokia OS, RIM OS, Android, BREW, etc.), API standards (MIDP 1.0, MIDP 2.0, etc.), optional/proprietary APIs, variations in accessing hardware (e.g., full screen support), maximum binary size allowed, etc.
  - **Implementation diversity** is caused by factors such as quirks/bugs in implementing standards.
- **Feature variations**, such as light version vs full version
- **User-preference diversity**, in aspects such as the language, style, etc., or accessibility requirements
- **Environmental diversity**, such as diversity in the deployment infrastructure (e.g., branding by carrier, compatibility requirements of the carrier's back-end APIs, etc.), locale, local standards.

As we can see from the above, one OC can differ from another due to many factors. Let us call these factors *fragmentors.* i.e., a fragmentor is a factor, diversity of which causes fragmentation. The fragmentation of mobile applications is often referred to as *device* fragmentation, because most of the fragmentors can be traced to a particular device model. This is a misnomer however, as factors outside the device (e.g., branding by carrier) too can cause fragmentation.

Since it is the diversity that drives fragmentation, a closer look at diversity may provide us with clues as to how to deal with fragmentation. It is our opinion that diversity can be either *essential* or *accidental*.[3]

- **Essential diversity** is the diversity that differentiates a product/service in some useful manner. Such diversity is intentional and often unavoidable. For example, users will continue to differ in their preferred size for a device, and the device manufacturers will continue to differentiate the devices in terms of size.
- **Accidental diversity** is the diversity that - does not serve any useful purpose, is often introduced unintentionally, and is often avoidable. For example, diversity due to API implementation bugs/quirks is unintentional, avoidable, and does not serve any useful purpose

Fragmentation is often associated with JavaME (Java Mobile Edition) applications, but it is also applicable to non-JavaME applications. Theoretically, a JavaME application is able to run on any Java-enabled mobile device. This means a JavaME application can target a much wider range of OCs as compared to non-Java applications, exposing it to more diversity. As non-JavaME applications (e.g., native applications for Symbian platform) are created for a smaller range of devices, they are exposed to less diversity. While a JavaME application has to run on platforms developed by many vendors, a typical non-JavaME application will run on a platform implemented by a single vendor or a small number of vendors (e.g., Symbian). This means JavaME applications have to face more implementation diversity, as compared to non-JavaME applications. However, developers may still have to develop a JavaME equivalent as well, if a wider range of OCs is to be targeted.

3. Effects of fragmentation

Fragmentation, and the subsequent de-fragmentation, complicates all disciplines[4] of a mobile application project. Some examples are given next.

- **Business modeling**: Business analysts have to determine the optimum set of OCs for the application to target. Questions to be answered include "Is operating context OC1 suitable for application A1?" and "Is it worth porting A1 to OC1?".
- **Requirements management**: If the interaction between the actor and the application is OC-dependent, it complicates the use-case specification by introducing a vast number of exceptional/alternate flows.
- **Analysis and design**: The system architecture, and the detailed design, should be able to accommodate the OCs targeted at the time, but also any future OCs the application will be exposed to during its lifetime.
- **Implementation**: Implementers need to optimize the application to all the targeted OCs. Questions to answer include "What do I have to do to fit application A1 to fit operating context OC1?", "How does OC1 differ from OC2?", and "Which OCs can be served by a single version of the application?"
- **Testing**: The application need to be tested for all targeted OCs. It is usually not enough to test on device emulators, as real devices on a real network sometimes behave differently from the emulators.
- **Project management**: Having to accommodate new (and unexpected) OCs in the middle of a project complicates project scheduling.
- **Configuration and change management**: Having multiple versions of an application (to suit multiple OCs) clearly impacts this discipline. New devices entering the market will increase the version count, while evolution of the platform software may require substantial changes to the existing versions.
- **Environment**: The software process has to be augmented to cater for additional complications

[3] This classification is borrowed from Fred Brooks' seminal book *The Mythical Man-Month*, which discusses "essential difficulties" and "accidental difficulties" of software development

[4] *disciplines* as defined in the IBM Rational Unified Process

introduced by fragmentation. For example, additional tools will be required to tackle various fragmentation issues.

Aforementioned complications increase the required effort in almost all aspects of the software life cycle, driving up the cost, and lengthening the time-to-market. Other side-effects are:

- It could reduce the quality of the product - The additional complexity of maintaining a large number of versions could increase the probability of bugs. Cost considerations may tempt developers to release applications that behave in sub-optimal ways for certain OCs (E.g., an application may work well for certain screen sizes, but may appear distorted in certain other screen sizes).
- It could narrow the target market - Cost considerations may force the application vendors to target a smaller market than the actual potential market it could target otherwise (see Figure 1).
- It hinders the growth of the mobile application market, by acting as a barrier-to-entry for new entrants - This is because creating a mobile application to fit a wide variety of OCs requires a much higher effort and a better expertise, when compared to a desktop/web application.

4. A taxonomy of de-fragmentation techniques

One way to reduce fragmentation is by eliminating diversity. However, only accidental diversity, which does not serve any useful purpose, should be targeted for elimination. Measures such as better standardization (e.g., less optional APIs, more detailed specifications), stricter enforcing of the standards (e.g., using API verification initiatives, Technology Compatibility Kits) can help in this regard. Major players in the mobile application industry such as platform vendors, device manufacturers, and carriers have a critical role to play in this front of the war against fragmentation. One such effort in the JavaME arena is the Mobile Service Architecture [7].

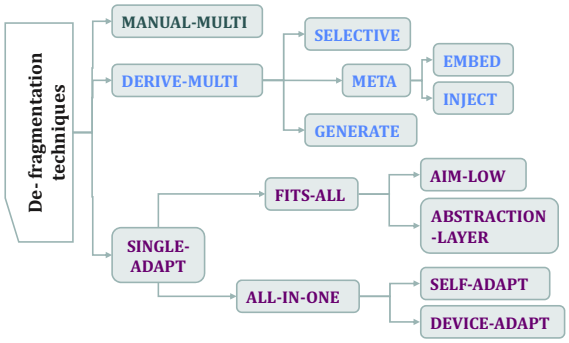


Figure 2. The complete taxonomy

On the other hand, essential diversity will be much harder, if not impossible, to avoid. The pragmatic response here is to find ways to reverse the resulting fragmentation. This is called *de-fragmentation* [3]. Note that de-fragmentation is NOT eliminating diversity. Rather, it is the process of making the application behave as intended on a set of diverse OCs.

In this section, we present a taxonomy of de-fragmentation techniques, based on the basic approach each technique uses. Figure 2 illustrates this taxonomy in its current state. Each approach will be explained in detail in the subsequent subsections. Note that a single application can use a combination of de-fragmentation techniques, using a different technique to manage each OC-specific variation.

4.1 The MANUAL-MULTI approach

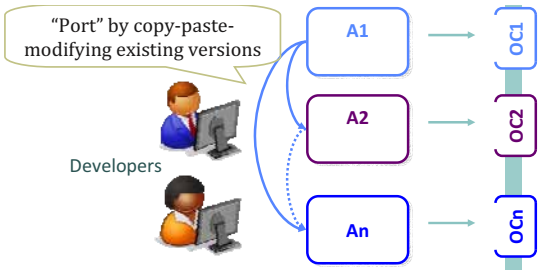


Figure 3. The MANUAL-MULTI approach

The most primitive way of de-fragmenting is to manually develop distinct versions of the application to suit different OCs. We call this approach *MANUAL-MULTI*. Figure 3 illustrates this approach, where *A1*, *A2*, … *An* are different versions of the application *A*, customized to fit operating contexts *OC1*, *OC2*, … *OCn* respectively. These distinct versions will be largely similar, but also different in subtle ways, as a result of subtle variations in the OCs. Copy-paste-modify techniques are commonly used to "port" the application to various OCs. MANUAL-MULTI approach results in duplication of work in many aspects of software development (e.g., fixing the same bug in hundreds of different versions). The following two alternative approaches try to minimize such duplication of efforts:

1. Derive OC-specific versions from a single code base (we call this approach *DERIVE-MULTI*)
2. Use a single version to serve multiple OCs (we call this approach *SINGLE-ADAPT*)

4.2 The DERIVE-MULTI approach

In the DERIVE-MULTI approach, we derive OC-specific versions of the application from a single code base. While this still results in multiple versions of the application, there is only one code base to work on and therefore, the effort required may be less than in the MANUAL-MULTI approach. In particular, we no longer need to manually maintain duplicate copies of the same source.

An example tool that supports the DERIVE-MULTI approach is the NetBeans Mobility Pack [8] (a JavaME mobile application development environment that comes as an extension to the popular NetBeans Java IDE). It uses a concept called *project configurations*, where a single application can have multiple project

configurations, one for each different versions we want to derive.

The DERIVE-MULT approach can be further sub-divided into three approaches: *SELECTIVE*, *META*, and *GENERATE*.

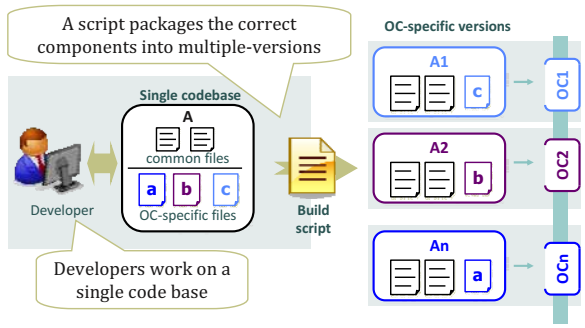


Figure 4. The SELECTIVE approach

The SELECTIVE approach (Figure 4) localizes variations into interchangeable components (e.g., classes, files, etc.) and uses a build script (or a linker) to create one version for each OC, picking out only the components required for that particular OC. This approach is frequently used when including images of different resolutions to fit different screen sizes. An example of this approach can be seen in the J2ME Polish tool [6]. For instance, we can put an image file in the *resources/ScreenSize.240+x320+* folder, and J2ME Polish will include this image for devices with a screen size of at least 240x320 pixels.

The META approach uses meta-programming (and similar code manipulation techniques) to *specify* how to derive OC-specific versions of the application. There are two ways of achieving this: the *EMBED* approach and the *INJECT* approach.

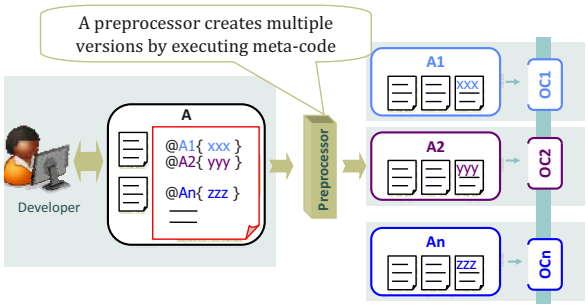


Figure 5. The EMBED approach

The EMBED approach embeds OC-specific variations in the source files using meta-programming directives/tags. A preprocessor derives multiple versions by processing these directives/tags. An example of this approach can be seen in NetBeans Mobility pack, which uses a concept called *preprocessor blocks* to specify OC-specific code segments. The example preprocessor block given in Figure 6 (adapted from [8]) is used to derive two different versions of the application, one for devices having 128x128 screens, and one for devices having 176x182 screens.

```
//#if screen == "128x128"
    //#  ballWidth = 10;
    //#elif screen == "176x182"
    //# ballWidth = 16;
//#endif
```

Figure 6. A NetBeans Mobility Pack preprocessor block

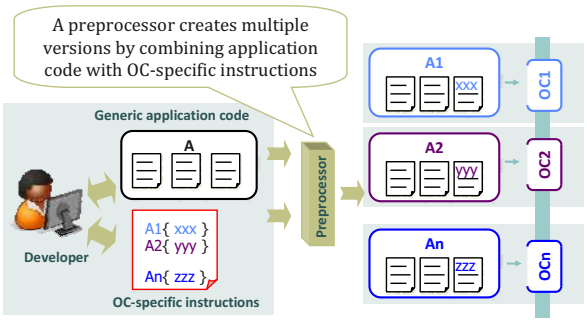


Figure 7. The INJECT approach

The INJECT approach requires the developer to write the OC-specific instructions separated from the application code. For example, Tira Jump [9] (a tool for developing mobile applications) uses aspect-oriented programming techniques to achieve such an effect. It lets developers write the application code against a reference OC and derives OC-specific versions by "weaving" OC-specific variations into it.

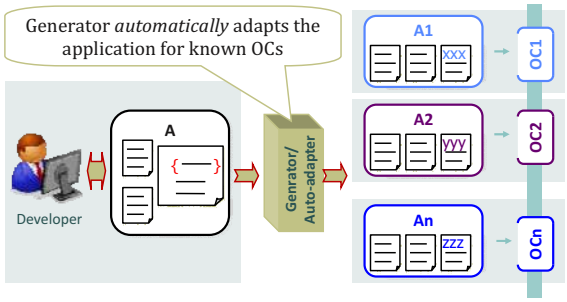


Figure 8. The GENERATE approach

The GENERATE approach automatically generates multiple versions using an intelligent generator that knows how to adapt a generic application to suit a specific OC. Instead of merely following instructions supplied by the programmer (as in the META approach), the generator uses its in-built knowledge in the generation process, requiring less manual coding. The feasibility of such fully automatic generation is rather limited, and we expect such generators to be limited to a narrow mobile application domain or a narrow range of OCs. For example, alcheMo tool [1] promises to automatically generate BREW format applications from JavaME applications.

4.3 The SINGLE-ADAPT approach

The SINGLE-ADAPT approach builds a single version of the application that can work on multiple OCs. This approach can be further sub-divided into two: *FITS-ALL* and *ALL-IN-ONE*.

The FITS-ALL approach develops a one-size-fits-all application that sidesteps all variations between OCs. There are two ways to accomplish this: *AIM-LOW* and *ABSTRACTION-LAYER*.

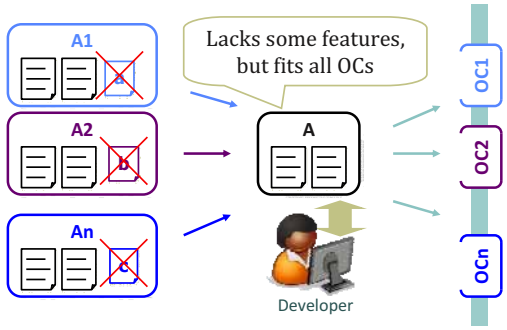


Figure 9. The AIM-LOW approach

The AIM-LOW approach (Figure 9) uses only what is common to all targeted OCs. For example, the UI will be designed to fit the smallest screen size of the targeted device range. This approach is sometimes referred to as the "lowest common denominator" approach.

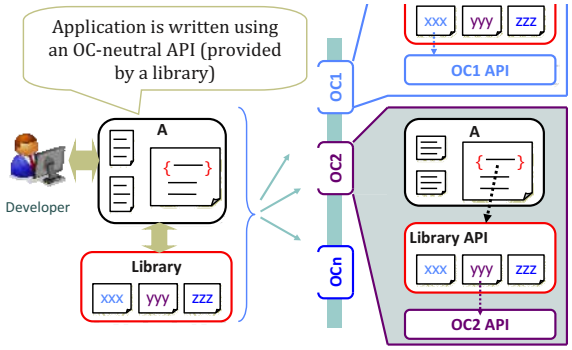


Figure 10. The ABSTRACTION-LAYER approach

The ABSTRACTION-LAYER approach (Figure 10), hides variations in the OCs behind an abstraction layer. This abstraction layer is usually a library (third-party or built in-house), and the application will be developed using the API of the library. Both the library and the application will be deployed on the mobile device, and it is the responsibility of the library to execute generic method calls from the application in an OC-specific manner. TWUIK [10] (a UI library for mobile applications) is one example tool that uses the ABSTRACTION-LAYER approach to write a single UIs that can adapt for multiple OCs.

The ALL-IN-ONE approach makes the software adapt at run-time to a given OC, using either the *SELF-ADAPT* approach or the *DEVICE-ADAPT* approach.

The SELF-ADAPT approach (Figure 11) makes the application programmatically discover *information* about the OC and adapt itself to the OC at run-time.

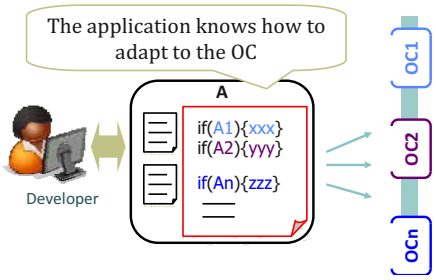


Figure 11. The SELF-ADAPT approach

In Figure 12 we see an example code snippet written in SELF-ADAPT fashion. This single piece of code will work for both screen sizes 128x128 and 176x182. The difference between this and the EMBED example in Figure 6 is that EMBED will include either `ballWidth=10;` or `ballWidth=16;` (but not both) in each OC-specific version, while SELF-ADAPT will include all code in Figure 12, resulting in a bigger application.

```
Canvas c = new Canvas();
w = c.getWidth (); h = c.getHeight();
if(w==128 && h==128)
    ballWidth=10;
 else if(w==176 && h==182)
    ballWidth=16;
```

Figure 12. An example of the SELF-ADPT approach

The DEVICE-ADAPT approach (Figure 13) requires the application to be written in an abstract way, and the device decides how to adapt it to the prevailing OC, at run-time. This approach is commonly used when dealing with fragmentation in the UI part of an application, often with unsatisfactory results. In Figure 14, we see how the same calculator application appears differently on two different phone emulators, after it has been adapted by the device.

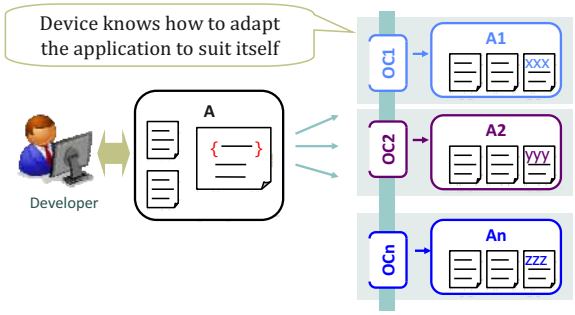


Figure 13. The DEVICE-ADAPT approach

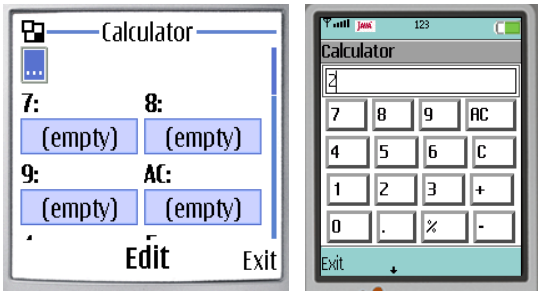


Figure 14. An example result from DEVICE-ADAPT

5. Related work

Fragmentation is one of the most talked about topics among practitioners (e.g., [3][4]). In academic research, fragmentation in the Mobile-Web has received frequent attention (e.g., [5]). Another related area is adaptable user interfaces. For example Gojas et al [2] describes a GENERATE type technique used to automatically generate UIs to fit different screens. The use of meta-programming to generate product lines is a well known technique, which could be adapted to de-fragment mobile applications. For example, Zhang and Jarzabek [11] shows how to use the XVCL meta-programming language (XVCL uses a combination of EMBED and INJECT approaches) to de-fragment a mobile game product line.

6. Conclusions and future work

In this paper, we analyzed the fragmentation problem faced by developers of mobile applications today. We defined the terms "operating context"- a concept central to the way we define fragmentation. We also explained our opinion of what it means to "de-fragment" an application, and contrasted it with eliminating diversity. As the major contribution of the paper, we presented a taxonomy of de-fragmentation techniques currently used in the industry, and used existing industry tools to illustrate each type of technique. Our future plans include a comprehensive evaluation of the techniques included in the taxonomy, to discover their strengths/weaknesses, to find synergies among them, and look for more effective alternatives. We shall continue to refine this taxonomy, based on interactions with the practitioners and our own experimentation.

7. Acknowledgements

Input from the following persons helped towards refining the material in this paper: Bhojan Anand, Naveed Shaikh and Nguyen Thi Tuyet Nhung (National Uni. of Singapore), Chris Abbott (DetectRight), Himath Dissanayake (OrangeHRM Inc), Kutila Gunasekera (Monash University), Jason Delport (Paxmodept), Luca Passani (WURLF) Mihai Fonoage (Florida Atlantic Uni.), Reto Senn (Bitforge), Ruchith Gunaratne (hSenid Software Intl) and Tom Hume (FuturePlatforms).

8. References

[1] alChemo home http://www.innaworks.com/alchemo

[2] Gajos, K, Christianson, D., Hoffmann, R., Shaked, T., Henning, K., Long, J. J., and Weld, D.S., "Fast And Robust Interface Generation for Ubiquitous Applications,". Proceedings of the Seventh International Conference on Ubiquitous Computing (UBICOMP'05), 2005

[3] JavaME: De-fragmentation Technical Overview and Design Guidelines Index, available at http://developers.sun.com/mobility/reference/techart/design_guidelines/overview.html

[4] Lau, A., " Fragmentation effect," http://www.javaworld.com/javaworld/jw-05-2004/jw-0524-fragment.html

[5] Liang, A., Guo, S., and Li, C., "Dynamic Mobile Content Adaptation Abstracting in Device Independent Web Engineering," Global Telecommunications Conference, 2006. (GLOBECOM '06), pp. 1 - 4

[6] J2ME Polish homepage http://www.j2mepolish.org

[7] JavaME mobile Service Architecture, http://java.sun.com/javame/technology/msa/

[8] Resolving JavaME Device Fragmentation Issues Using NetBeans 6.0 Mobility http://www.netbeans.org/kb/60/mobility/javame-devicefragmentation.html

[9] Tira Jump home page http://www.tirawireless.com

[10] TWUIK homepage http://www.tricastmedia.com/twuik/

[11] Zhang, W. and Jarzabek, S. "Reuse without Compromising Performance: Experience from RPG Software Product Line for Mobile Devices," *9th Int. Software Product Line Conference, SPLC'05*, September 2005, Rennes, France, pp. 57-69

Identifying NFRs Conflicts Using Quality Ontologies

Taiseera Al Balushi
Information Systems Department, College of Commerce and Economics, Sultan Qaboos University,
P.O. Box 20, PC 123 Al Khod, Oman
+968 24142970
taisira@squ.edu.om

Pedro R. Falcone Sampaio, Mitul Patel
Business Systems Division, Manchester Business School, The University of Manchester, Booth Street West, Manchester M15 6PB, UK
+44 (0)161 306 3349
P.Sampaio@manchester.ac.uk
Mitul.Patel@student.manchester.ac.uk

Oscar Corcho
School of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK
+44 (0) 161 275 6821
Oscar.Corcho@manchester.ac.uk

Pericles Loucopoulos
Business School, Loughborough University, Loughborough LE11 3TU, UK
+44 (0)1509 22 8273
P.Loucopoulos@lboro.ac.uk

Abstract

Conflict identification and resolution is a key phase of requirements engineering. It is crucial to identify conflicts at early stages of the requirements engineering which in turns helps in establishing a cohesive set of requirements to guide the overall requirements engineering process. Conflicts especially arise due to the self reinforcing or contradictory nature of some NFRs (e.g. efficiency and usability). This paper describes how quality ontologies can be used to support the identification of NFR conflicts and facilitate discussion towards requirements prioritization tasks in requirements engineering. Our approach is based on using the ISO/IEC 9126 quality ontology to underpin the NFR description and reasoning mechanisms to pinpoint potential NFR conflicts that need to be further discussed by stakeholders. The work is implemented in the ElicitO requirements elicitation tool. We also report results of applying the approach and the tool to identify conflicts in requirements elicitation activities at the student intranet project of the University of Manchester (Manchester Unity Web Project).

Keywords: Non-functional requirements, requirements engineering, conflict identification, ontologies.

1. Introduction

Addressing Non-Functional Requirements (NFRs) or Quality Requirements are vital to the success of software systems [1], playing a crucial role during systems development and serving as quality criteria for assessing software effectiveness [2]. Errors related to identification of NFRs are generally acknowledged to be the most expensive and difficult to correct once the information system has been completed [2, 3]. Without a well defined set of NFRs and their proper fulfillment, software projects are vulnerable to failure [4].

Thus, finding the right configuration of NFRs is an important step towards achieving a successful software deliverable. NFRs, on the other hand, have numerous complex and nontrivial interdependencies. NFRs conflict with each other when they make contradicting statements about a software attribute, and they cooperate when they mutually enforce such attributes [5]. As requirements are being elicited and modeled, the challenging task is to maintain an agreement between all the stakeholders. This is because it is common for conflicts to arise in connection to NFRs which often take place especially in a situation where there is a large number of stakeholders with different backgrounds and perceptions of the problem [6].

This paper describes how quality ontologies can be used to support the identification of NFR conflicts and facilitate discussion towards requirements prioritization tasks in requirements engineering. Our approach is based on using the ISO/IEC 9126 quality ontology to underpin the NFR description and reasoning mechanisms to pinpoint potential NFR conflicts that need to be further discussed by stakeholders. The work is implemented in the ElicitO requirements elicitation tool. We also report results of applying the approach and the tool to identify conflicts in requirements elicitation activities at the student intranet project of the University of Manchester (Manchester Unity Web Project).

The remainder of this paper is divided as follows: Section 2 discusses issues related to conflict identification. Section 3 provides examples of conflicts in quality requirements. Section 4 discusses how quality ontologies are used to support requirements elicitation and conflict identification. Section 5 describes the design of the ontology for conflict identification. Section 6, provides an example using ElicitO tool to identify conflicts and section 7 presents some related work in conflict identification. Section 8, summarizes the paper, discusses the key contribution, and future work.

2. Conflict Identification

The term conflict can be taken to mean interference in one's party's activities, needs or goals, caused by the activities of another party [7]. Literature concerned about conflicts originates from different fields such as social psychology, cognitive science, and sociology [8]. However and for the purpose of this paper we will focus on conflicts in the requirements engineering literature which is defined by Lamsweerde [9] as "conflict is a divergence between goals – there are feasible boundary conditions that makes the goals inconsistent". Robinson [10] also argued that many inconsistencies originate from conflicting goals; inconsistency management should, therefore, proceed at the goal level.

Easterbrook [7] identified two sources of conflicts in requirements engineering: conflicts between the participants perceptions of the problems, and conflicts between the many goals of design. Conflicts can also arise in connection to NFRs, thus NFRs can make conflicts and cooperation instances more obvious, because changes in quality attributes often cause certain functional changes that in turn affect other NFRs [5].

Many methods exist to deal with conflict resolution in requirements but for the purpose of this paper we will investigate the approaches that deal with NFR conflict resolution. McCall [11] provided a checklist of attribute capabilities to be considered in requirements specifications without an automated conflict analysis. The NFR-goal framework [12] views NFRs as goals that might conflict with each other and they must be represented as softgoals to be satisfied, this is achieved by propagating such information along positive/negative support links in the goal graph. Boehm and In [4] propose a knowledge base where NFRs are prioritized through the stakeholders' perspective, dealing with NFRs high level of abstraction. Easterbrook [7] provides a framework for conflict resolution between domain specifications. Egyed [5] identifies requirements conflicts and cooperation using software attributes and eliminates false conflicts and cooperation automatically with the help of a trace analysis technique.

Although these approaches analyzed the identification, communication, and conflict resolution, they do not comprehensively address the following issues:

- Define a terminology for standardizing the non-functional requirements definitions and meanings.
- Examine the nature and correlations between NFRs that potentially may result in a conflict.
- Automate the process of conflict identification using knowledge management techniques.

This paper describes how quality ontologies can be used to support the identification of NFR conflicts and facilitate discussion towards requirements prioritization tasks in requirements engineering. Our approach is based on using the ISO/IEC 9126 quality ontology to underpin the NFR description and reasoning mechanisms to pinpoint potential NFR conflicts that need to be further discussed by stakeholders.

3. Examples Of Conflicts In Quality Requirements

NFRs, as investigated by [5], conflict with each other when they make contradicting statements about some software attribute, and they cooperate when they mutually enforce such attributes. Thus it is important to understand how NFRs relate to each other in order to identify the key conflicts early in the requirements elicitation process and before the project evolves to a situation where it is hard to manage the set of NFRs developed by the stakeholders.

In this section we study the relationship among NFRs. We adopted the ISO/IEC 9126 [13] as a standard quality model and terminology for describing NFRs. Table 1 shows some relationships between quality requirements at the quality sub-characteristics level. The quality requirements may cooperate (+), conflict (-), or have no effect with each other (0). This model was adopted from a range of contributions specialized in analyzing the quality requirements relationships [5, 14, 15]. The table does not cover all the quality sub-characteristics listed in the ISO/IEC 9126 limiting it to the common ones used in software projects; usability [16-19], security [20] and efficiency [21].

Table 1 Correlations between ISO/IEC 9126 Quality Requirements

Quality Requirement	Effect										
	Accuracy	Interoperability	Security	Recoverability	Fault Tolerance	Learnability	Understandability	Attractiveness	Operability	Time Behaviour	Resource Utilization
Accuracy	+	0	0	0	0	+	+	+	+	-	-
Interoperablity	0	+	0	0	0	0	0	0	+	0	-
Security	+	0	+	0	0	0	0	-	-	-	-
Recoverability	0	0	0	+	+	0	0	0	+	-	-
Fault Tolerance	0	0	0	+	+	0	0	0	+	-	-
Learnability	+	0	0	0	0	+	+	0	+	0	0
Understandability	+	0	0	0	0	+	+	0	+	0	0
Attractiveness	0	+	-	0	0	+	+	+	+	0	0
Operability	+	+	-	+	+	+	+	+	+	+	+
Time Behaviour	-	0	-	-	-	0	0	0	+	+	-
Resource Utilizatio	-	-	0	-	-	0	0	0	0	-	+

(+) represents a positive effect, (-) represent negative effect, (0) represents no effect

In an ideal universe, every system would exhibit the maximum possible value for all its quality requirements but this is often unattainable. Thus it is important to learn which quality requirements are most important to the success of a project. From Table 1, design approaches that require higher accuracy also enforce other quality requirement such as learnability, understandability, attractiveness, and operability. However, higher accuracy may also increase response time and resource consumption which are often undesirable by stakeholders. Therefore, to reach the optimum balance of quality requirements, we must identify, specify, and prioritize the pertinent quality attributes during requirements elicitation.

4. Using Quality Ontologies To Support Elicitation And Conflict Identification

Quality ontologies were used with requirements elicitation by providing the requirements analysts with the knowledge repository to support elicitation activities [22, 23] by defining quality sub-characteristics and metrics that need to be specified towards describing the requirements with appropriate levels of precision. Quality ontologies can also be used to support conflict identification in connection to NFRs by offering the following benefits:

- Provide a shared domain vocabulary for the NFRs to avoid ambiguities among stakeholders.
- Analyze the relationships between quality requirements in order to avoid combining conflicting requirements by stakeholders.

- Encode specialized knowledge to support the formulation of competency questions with regard to quality requirements meanings and relationships among each other. Thus facilitating the elicitation of a complete set of conflict free quality requirements.

In order to achieve these goals, our motivation is to develop an ontology driven requirements elicitation and negotiation/prioritization method, guided by a standard quality model. The quality model is encoded as a quality ontology, and automated by a requirements elicitation tool ElicitO[23], helping to address quality factors during elicitation interviews as well as dealing with NFRs trade-offs. Figure 1 illustrates the proposed approach. There are two main ontologies important to guide the elicitation and conflict identification: *Quality ontology*, which is based on software quality models representing reusable knowledge about different quality characteristics, sub-characteristics, and metrics. *Domain ontology*, which provides a conceptual structure of the domain (e.g. university helpdesk, in this paper) including functions, activities, relationships, etc.

The implementation of the ElicitO tool [23] was carried out using Protégé. It only addresses requirements elicitation by empowering requirements analysts with expert domain knowledge about the functional aspects via a domain ontology and non-functional requirements via a quality ontology relevant to a given domain. For the purpose of this paper we continue working with the quality ontologies to help with requirements negotiation and conflict identification once the requirements are elicited.

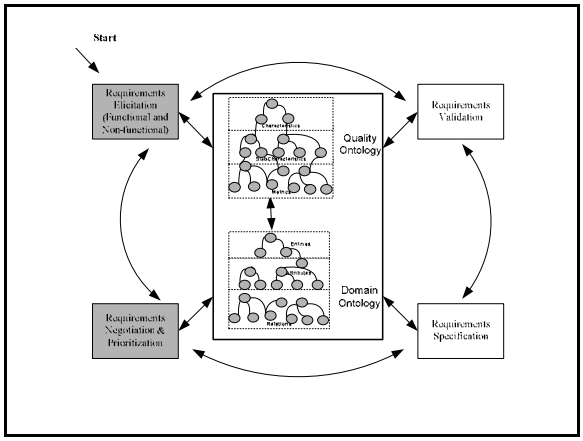


Figure 1 Ontology Guided Requirements Elicitation and Conflict Identification Framework

As the requirements are being elicited [22] the requirements analyst can also assist the stakeholders in analyzing and validating these sets of requirements for identifying conflicts. The ElicitO tool facilitates this process by highlighting potential conflicts to allow further communication and negotiation until the stakeholders' reach an agreement and quality attributes are prioritized.

5. The Design Of The Ontology For Conflict Identification

The ontologies for conflict identification are developed in OWL and they describe the domain classes, properties and restrictions of the functional and quality requirements knowledge, as illustrated Figure 2. There are two ontologies underpinning the conflict identification process:

- Quality Ontology: represent the quality taxonomy which is decomposed into four main components as shown in Figure 2 (1) and corresponds to the relationships between quality characteristics (Conflict or influence each other), (2) corresponds to the ISO/9126 quality model (quality characteristics and quality sub-characteristics), and (3) represents the quality metrics. The quantitative measures for the metrics are borrowed from SUMO [24] (information, time, length, and mass measures).
- Domain Ontology, and in our case is the helpdesk ontology for which we used text books, standards, and interviewed domain experts (helpdesk operators with more than 5 years of experience each). We have also borrowed some classes and properties defined in other ontologies such as SUMO [24]. For examples SUMO Entity (page, center, helpdesk, student), SUMO processes (borrowing, search, register) etc.

The restrictions are then defined for classes in the previous ontologies to determine what metrics are representing the quality characteristics and sub-characteristics as shown in Figure 2. In addition, it represents the metrics related to the domain activities.

The restrictions specified above were used to restrict an individual that belongs to a class (e.g. helpdesk *has-metric* page_downloads_speed). The quality ontology, however, doesn't provide a mean of performing specific actions on the ontologies (i.e. conflict identification). In order to conduct conflict identification actions we incorporated conflict identification reasoning to the system by applying the rule reasoning framework supported by JessTab [25]. Although the rules are expressed in Jess, other languages such as SWRL (Semantic Web Rule Language) could be used; however we have selected Jess due to its configurability and usability in protégé via Jesstab.

There is a rule for each pair of quality attributes as indicated in Table 1. These rules are to be fired when the stakeholders combine two conflicting requirements in order to alert the stakeholders and allow further discussion. Examples of Jess Rule that will be fired when the stakeholders combine two conflicting requirements are shown in Figure 3.

The ElicitO tool also offers additional features using Jess Rules such as:

- Separation between the knowledge base model (quality and domain ontologies) and the model where the actions are performed. This is because the first model is standardized and shared with regards to quality attributes related to a particular domain, however, the second model reuses the first model but with extra actions depending on the objectives of the ontology based applications, in our case it's used for requirements elicitation and conflicts identification.

- Help with identifying conflicts early as requirements are elicited to facilitate further discussion among stakeholders until they reach an agreement and prioritize requirements.

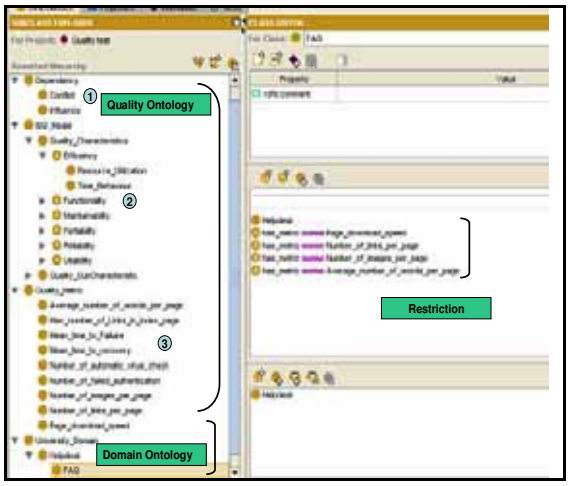


Figure 2 Quality Ontologies in Protégé

By extending the features of ElicitO with conflict identification capabilities, the requirements analysts are empowered with a knowledge repository to help with requirements elicitation and conflict identification. The automation of conflicts identification tasks is especially useful in projects involving multiple stakeholders and that can scale up to thousands of requirements.

6. Running Example Using ElicitO

To assess the effectiveness of the approach, the authors attended a focus group session which was one of the ongoing sessions in connection with University of Manchester Unity Web Project for the purpose of enhancing the current helpdesk website of the university. The participants were from different departments with different views, assumptions, and as a result, different requirements. The participants were asked for what they want to see in the new system and what sort of problems they have encountered with the old system. A two hour session was conducted jointly with stakeholders, the first hour was dedicated to requirements elicitation and the second hour was dedicated to requirements prioritization. The requirements elicited during the elicitation phase are as presented in Table 2. As indicated from the table, the types of requirements are limited, very general, and vary from functional and non-functional requirements with very little attention to quality requirements (R2, R3, and R5).

The second stage of the session was the requirements prioritization on which the requirements engineer asked the participants to rank the above requirements with either *essential* or *nice to have* as illustrated in Table 3. This had the potential to trigger conflicts as every participant would vote high for what they want disregarding how their requirements might conflict with others. For example, R5 and R6 are considered essential by the majority of the participants but they might conflict with the issue of security which wasn't taken into account by the requirements analysts.

```
(defrule time-security (Req_Time_Behaviour TRUE)
  (Req_Security TRUE) =>  (printout t "Time Behaviour and
  Security are Conflicting requirements" crlf))
(defrule attractivness-security (Req_Attractiveness TRUE)
  (Req_Security TRUE) => (printout t "Attractiveness and
   Security are Conflicting requirements" crlf))
```

Figure 3 Jess Rules Example

In contrast to the unstructured and ad-hoc approach conducted during the focus group sessions, another session was conducted using the ElicitO tool for the requirements elicitation and prioritization activities which provided the relevant domain and quality knowledge to the requirements analysts to be more effective in conducting the elicitation/prioritization interviews. The tool highlights all the functional activities of the domain and their attached quality characteristics. The analyze requirements button will examine the requirements for potential conflict anytime during the requirements elicitation Figure 4(a). The analyze requirements button will fire the Jess Rule that will check the requirements for potential conflicts and a list of conflicting requirements are as highlighted in Figure 4 (b). The analyst then selects a set of conflicting requirements to allow further discussion/negotiation and prioritization Figure 4 (c).

Table 2 Requirements Captured without the tool support

	User Requirements
R1	Provide information/pathway onto how to access web services (i.e. web mail, network drive, etc.)
R2	FAQ should be clear and simple in answering users technical problems
R3	Make the websites among different schools consistent
R4	Provide campus map when required
R5	Make the university regulations and policies easy to access
R6	Make students user names accessible to faculty when using WebCT (e-learning) to register students
R7	Provide information on how to report a problem and to whom
R8	Provide information about exam timetables and venues
R9	Provide links to the outside world
R10	Highlight important events or alerts
R11	Update the staff directory frequently

For the discussion and prioritization activity, all the participants assess the perceived return on value of the quality requirement by each participant using a scale from (1-5): 1-no value, 2-little value, 3-some value, 4-high value, 5-very high value [26]. For each requirement the mean value of all participants' assessment is calculated and a priority is specified. The participants can also write a short justification for choosing a certain quality requirement over the other. The same process is applied for each conflicting requirements. Figure 4 (b) presents the prioritized requirement of one quality requirement

over the other requirement in addition to the detailed requirements specifications (using the same amount of time as of the first session).

Table 3 Requirements Prioritized without the tool support

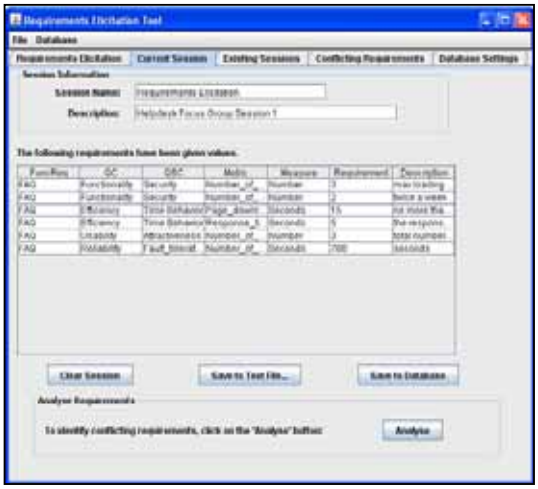
	User Requirements	Priority
R1	Provide information/pathway onto how to access web services (i.e. web mail, network drive, etc.)	Nice to have
R2	FAQ should be clear and simple in answering users technical problems	Essential
R3	Make the websites among different schools consistent	Nice to have
R4	Provide campus map when required	Essential
R5	Make the university regulations and policies easy to access	Essential
R6	Make students user names accessible to faculty when using WebCT (e-learning) to register students	Essential
R7	Provide information on how to report a problem and to whom	Nice to have
R8	Provide information about exam timetables and venues	Essential
R9	Provide links to the outside world	Nice to have
R10	Highlight important events or alerts	Nice to have
R11	Update the staff directory frequently	Essential

The findings obtained from the focus group sessions with ElicitO support can be listed up as follows:

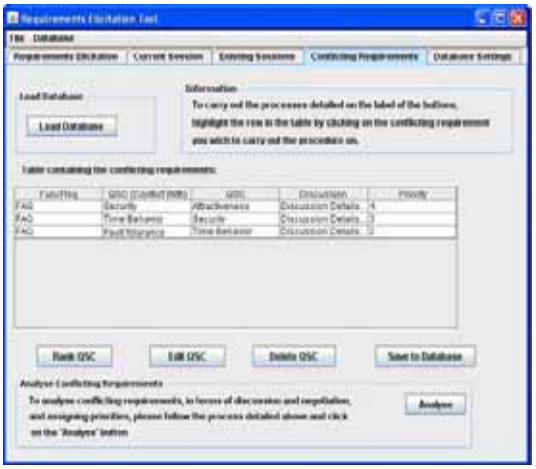
- The knowledge encoded in the ontology formalizes the quality requirements and makes them explicit throughout the requirements elicitation process which reduces the problem of understanding caused by different interpretations of quality requirements.
- The knowledge encoded in the ontology is based on the ISO/IEC 9126. Quality model extended by adding metrics and defined relationships among the quality factors to enable analysts in capturing a rich set of non-functional requirements.
- The numbers of functional and quality requirements captured were far more than the initial number of requirements elicited without the tool support. The quality requirements were associated with the functional requirements which have added value to the functional requirements.
- The non-functional requirements were not only extensively identified by the stakeholders but they were also precisely specified via metrics.
- The tool identifies the conflicting requirements early in the process so the stakeholders can negotiate and rank the requirements. Thus facilitating and speeding up the software engineering process.

Overall the ElicitO tool facilitated the requirements elicitation activities by providing the required functional requirements, quality requirements and precise metrics to the requirements analysts about a specific application domain via the knowledge encoded in the ontology. ElicitO also helped with the identification of potential conflicts among desired quality attributes and facilitated agreement on a balance of attribute satisfaction via communication and quality requirements prioritization.

(a)



(b)



(c)

Figure 4 (a): requirements document; (b): list of conflicting requirements; (c): conflicting requirements negotiation/prioritization

7. Related Work

In general, our approach complements the other work related to quality requirements conflicts identification. Boehm and In [4] proposed Quality Attribute Risk and Conflict

Consultant knowledge-base tool (QARCC) an exploratory knowledge-based tool for identifying potential conflicts and risks among quality requirements early in the software life cycle. QARCC uses a knowledge base to identify software architecture and process strategies to achieve this quality attribute. Another approach is the requirements negotiation tool (Oz) [27] which effectively support an automated conflict detection, characterization, and resolution generation, and resolution decision-making support. In the NFR framework [12] quality requirements are identified, decomposed, and prioritized so an effective design solution is found. Our proposed method ElicitO, improved on the other approaches by supporting the quality requirements elicitation and conflict identification for both functional and non-functional requirements via quality ontology knowledge based domain independent tool.

8. Conclusions And Future Work

This paper proposes an elicitation and conflict identification approach for non-functional requirements and associated tool ElicitO aimed at supporting requirements analysts with a knowledge repository that helps in eliciting a comprehensive and conflict free set of requirements. The approach is based on the application of functional and non-functional domain ontologies (quality ontologies) to underpin the elicitation and conflict identification activities.

The ISO/IEC 9126 quality model was adopted as a baseline for addressing quality concerns and the NFRs relationships are analyzed and codified using rules to help with reasoning about conflict identification. The approach and the tool were evaluated using a web project at the University of Manchester, where it proved to help in identifying potential conflicts and allowing participants to further discuss the requirements to effectively and efficiently reach an agreement.

9. References

[1] L. Chung and B. A. Nixon, "Dealing with non-functional requirements: three experimental studies of a process-oriented approach," presented at Proceedings of the 17th international conference on Software engineering, Seattle, Washington, United States, 1995.

[2] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and Using Non-Functional Requirements: A Process Oriented Approach," *IEEE Transactions on Software Engineering*, vol. 18, pp. 483-497, 1992.

[3] L. M. Cysneiros and J. C. S. d. P. Leite, "Integrating Non-Functional Requirements into Data Modeling," presented at Proceedings of IEEE International Symposium on Requirements Engineering, Ireland, 1999.

[4] B. Boehm and H. In, "Identifying quality-requirement conflicts," *Software, IEEE*, vol. 13, pp. 25-35, 1996.

[5] A. Egyed and P. Grunbacher, "Identifying requirements conflicts and cooperation: how quality attributes and automated traceability can help," *Software, IEEE*, vol. 21, pp. 50-58, 2004.

[6] B. Nuseibeh and S. Easterbrook, "Requirements Engineering: A Roadmap," presented at Proceedings of the conference on The future of Software engineering, Limerick, Ireland, 2000.

[7] S. Easterbrook, "Resolving Requirements Conflicts with Computer-Supported Negotiation," in *Social and Technological Issues in Requirements Engineering*, M. Bickerton and M. Jirotka, Eds.: Academic Press, 1993.

[8] S. Easterbrook, E. Beck, S. Goodlet, L. Plowman, M. Sharples, and C. Wood, "A survey of empirical studies of conflict," in *CSCW: Cooperation or conflict?*: Springer-Verlag, 1993, pp. 1-68.

[9] A. Lamsweerde, E. Letier, and R. Darimont, "Managing Conflicts in Goal-Driven Requirements Engineering," *IEEE Transactions on Software Engineering*, vol. 24, pp. 908 - 926, 1998.

[10] W. N. Robinson, "Integrating Multiple Specifications Using Domain Goals," presented at Proc. IWSSD-5—Fifth Int'l Workshop Software Specification and Design, Pittsburgh, United States, 1989.

[11] J. A. McCall, P. K. Richards, and W. G.F, "Factors in Software Quality," Technical Report, AD/A-049-014/015/055, National Technical Information Service 1977.

[12] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Norwell, Massachusetts: Kluwer Academic Publishing, 2000.

[13] "ISO/IEC 9126-1:2001 Software engineering --Product quality -- Part 1: Quality model."

[14] X. Franch and J. P. Carvallo, "Using quality models in software package selection," *IEEE Software*, vol. 20, pp. 34-41, 2003.

[15] K. Wiegers, *Software Requirements: Practical Technologies for Gathering and Managing Requirements Throughout the Product Development Cycle*: Redmond, Wash- Microsoft Corp, 2003.

[16] J. Nielsen, *Designing Web Usability: the practice of simplicity*: New Riders Publishing, 1999.

[17] A. Abran, A. Khelifi, W. Suryn, and A. Seffah, "Usability Meanings and Interpretations in ISO Standards," *Software Quality Journal*, vol. 11, pp. 325–338, 2003.

[18] L. M. Cysneiros, V. M. Werneck, and A. Kushniruk, "Reusable Knowledge for Satisficing Usability Requirements," presented at Proceedings of the13th IEEE International Conference on Requirements Engineering, 2005., 2005.

[19] M. Moraga, C. Calero, I. Paz, O. Dجaz, and M. Piattini, " A Reusability Model for Portlets," presented at International Workshops on Web Information Systems Engineering, WISE 2005, New York, USA, 2005.

[20] D. Firesmith, "Engineering Security Requirements," *Journal of Object Technology*, vol. 2, pp. 53-68, 2003.

[21] Y. Yuan, "Efficiency metrics model for component-based embedded application software," presented at Second International Conference on Embedded Software and Systems, 2005., 2005.

[22] T. AlBalushi, P. Sampaio, D. Dabhi, and P. Loucopoulos, "Performing Requirements Elicitation Activities Supported by Quality Ontologies," presented at Proceedings of the Eighteenth International Conference on Software Engineering and Knowledge Engineering, San Francisco, 2006.

[23] T. AlBalushi, P. Sampaio, D. Dabhi, and P. Loucopoulos, "ElicitO: A Quality Ontology-Guided NFR Elicitation Tool," presented at Proceedings of the 13th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'07), Trondheim, 2007.

[24] SUMO, "Standard Upper Merged Ontology," vol. April, 2007, 2003.

[25] H. Eriksson, "Using JessTab to integrate Protege and Jess," *IEEE Intelligent Systems*, vol. 18, pp. 43-50, 2003.

[26] S. Ziemer, P. R. F. Sampaio, and T. Stهlhane, "A decision modelling approach for analysing requirements configuration trade-offs in time-constrained Web Application Development," presented at Proc. Eighteenth International Conference on Software Engineering and Knowledge Engineering (SEKE 2006), San Fransisco, 2006.

[27] W. N. Robinson and S. Fickas, "Automated Support for Requirements Negotiation," presented at AAAI-94 Workshop on Models of Conflict Management in Cooperative Problem Solving, 1994.

Ontology-Based Process Modeling and Execution using STEP/EXPRESS*

Arndt Mühlenfeld, Wolfgang Mayer, Franz Maier, Markus Stumptner

Advanced Computing Research Centre
University of South Australia
5095 Mawson Lakes, SA, Australia
E-mail: {muehlenfeld|wolfgang.mayer|franz.maier|mst}@cs.unisa.edu.au

Abstract

A common data format as provided by the STEP/EXPRESS initiative is an important step toward interoperability in heterogeneous design and manufacturing environments. Ontologies further support integration by providing an explicit formalism of process and design knowledge, thereby enabling semantic integration and re-use of process-information. By formalizing the process-model in EXPRESS, we gain access to the domain knowledge in the STEP application protocols. We present an approach to process modeling using different models for abstract process knowledge and implementation details. The abstract process model supports re-use and is independent of the implementation. As a result, we translate the process model in combination with the implementation model to an executable workflow.

1. Introduction

Modern industrial design manufacturing processes allow for collaboration among organizations and organizational units within large companies. Design knowledge that is spread over several design teams and systems is difficult to integrate. The lack of interoperability in heterogeneous information systems results from incompatible data formats and differences between domain models. Data exchange between design stages requires definition of mappings between data representations or the use of a common format. STEP/EXPRESS is an established standard for product data representation and solves the problem of incompatible data formats. Differences in domain models are addressed by modeling semantic knowledge in ontologies. Semantic interoperability between design disciplines is usually achieved by using a common upper ontology [12] or mappings between domain ontologies [16]. N.Guarino [4] proposes a model for information integration that uses separate ontologies for task and domain knowledge with a common upper ontology.

In this paper we present a meta-model for task ontologies of industrial processes that integrates process knowledge with artifact representation. The meta-model provides us with the means to express knowledge over artifacts and reconstruct provenance. We utilize established workflow execution engines for enacting the process model by building a meta-model for the execution environment and defining a mapping between the process and the enactment meta-models. The process meta-model has two parts, an abstract *process model* and an *implementation model*. Keeping the implementation details separate from the process model makes the conceptual model clearer and better suited for re-use. We use EXPRESS to specify our ontological model, which gives us direct access to the information models of the STEP Standard. Furthermore, we can use the same language for (a) process models and (b) artifacts represented in STEP. In order to close the gap between specification and implementation we present a mapping of process specifications to a specific workflow engine. Most workflow engines provide an execution trace of the enacted workflow and support data provenance. We present an example mapping for a specific workflow execution engine to demonstrate that our process model contains the necessary information. The process model is independent of the actual workflow engine and can be mapped to several different engines. Specification of mappings between our process meta-model and the meta-model of a specific workflow engine enables automatic translation of process models. Hence, we are free to use the workflow execution engine that best suits the target environment.

In Section 2 we introduce our meta-model and the benefits arising from formalizing it using STEP/EXPRESS. Section 3 is dedicated to the enactment of the process model. We use an implementation model and transformations to create a specific workflow that can be fed to a workflow

*This work was funded by the CRC for Advanced Automotive Technology under project C4-801 Process Modelling in the Automotive Industry.

execution engine. An example of a process-model transformation is given in Section 4. In Section 5 we present related work. Our contribution and future work is summarized in Section 6.

2. Process Modeling with EXPRESS

2.1. The EXPRESS Language

The STEP standard (ISO 10303) defines a collection of application protocols representing data models for different domains. EXPRESS is the modeling language used for the data models and is specified in Part 11 of the standard [6]. The language is able to represent entity-relationship concepts in an object-oriented way. Its powerful representation of constraints on data has shown to be suitable for formal specifications and meta programming [1]. An application model in EXPRESS comprises types, functions and data objects called *Entities*. Entities consist of attributes and constraints related to the attributes. Entities are the central elements of the language and represent classes of objects. As in object-oriented languages, classes are structured hierarchically by inheritance. The elements of a model are grouped into a *Schema*. Schemata are like name spaces and can be referenced by other schemata.

The language is powerful enough to express the structure of any meta-model within an EXPRESS Schema and the standardized access interface in several languages bindings allows for the generation of a meta-data management systems suited to the target systems [14].

2.2. The Process Model

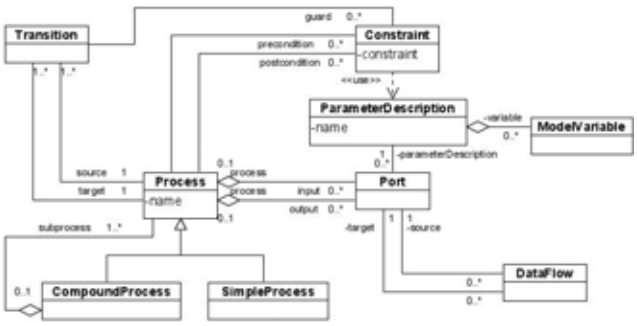


Figure 1. An overview of the meta-model in UML.

In the meta-model depicted in Figure 1 an abstract *Process* is either a *CompoundProcess* comprising one or more processes or an indivisible *SimpleProcess*. A Process has input and output *Ports* for input and output data. The data expected on a port is specified by a *ParameterDescription* which in turn includes *ModelVariables*, if it represents a complex structure like a parametric model. *Constraints* specify the behavior of the process in the role of preconditions and postconditions. A *DataFlow* connects output ports with input ports and is the basic building block for data flow in the process model. Control flow is modeled independently of the data flow by *Transitions*. A process can have many ingoing and outgoing transitions.

```
ENTITY ParameterDescription;
    description : STRING;
    variable : SET [ 0 : ? ] OF ModelVariable;
    is_similar_to : SET [ 0 : ? ] OF ParameterDescription;
END_ENTITY;

ENTITY Port
    SUBTYPE OF ( NamedEntity );
    descr : ParameterDescription;
END_ENTITY;

ENTITY DataFlow;
    source : Port;
    target : Port;
WHERE
    data_is_compatible :
        ( source.descr = target.descr ) OR
        ( source.descr IN target.descr.is_similar_to ) OR
        ( target.descr IN source.descr.is_similar_to );
END_ENTITY;
```

Listing 1. Specification of entities ParameterDescription and DataFlow in EXPRESS

The process model presented above does not specify the properties of the data exchanged between processes. The data model makes use of the application protocols of the STEP Standard and is domain dependent. We use AP 214 to model the data of the design optimization process, because it contains the domain knowledge for automotive design processes. The full data model is beyond the scope of this document. However, as an example for the expressiveness of EXPRESS, we have formulated a rule that ensures that *Dataflows* connect only *Ports* with "related" parameter descriptions. Therefore, we define the attribute *is_similar_to* in *ParameterDescription*, which represents the association to related objects (see Listing 1). Entity *DataFlow* has a rule stating that only ports that have the same parameter description, or parameter descriptions that are similar to each other, are allowed as source and target objects.

2.3. The Implementation Model

The process meta-model describes abstract properties, process components and their relationships, but not how a process can be executed or where the data for its ports are stored. This information is part of the implementation model. The implementation meta-model in Figure 2 defines two types of process instances, *WebService* and *Executable*. For the sake of brevity, details on Web Services are omitted. An *Executable* has at least two possibilities for its input. In

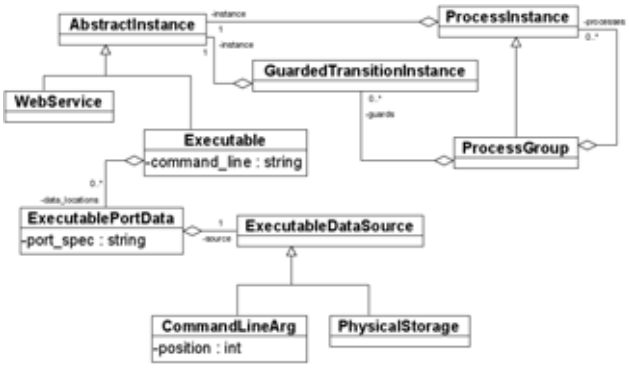


Figure 2. An overview of the implementation model in UML.

our model, it can receive input from physical storage (*PhysicalStorage*), e.g., a File, or as a command-line parameter (*CommandLineArg*). Physical processes are part of a *ProcessInstance* or a *GuardedTransitionInstance*. The former is the realization of a *Process* in the process-model; the latter is a process that evaluates the guards of a *Transition* in the process-model. Realizing the evaluation of guards for conditional execution as a physical process keeps the model independent of the constraint language. Not visible in the UML-diagram, but specified in EXPRESS is the constraint *has_result*. It expresses the invariant that an entity of class *GuardedTransitionInstance* has to provide a port named 'result' for the evaluation result (see Listing 2). This is possible, because *AbstractInstance* provides an attribute containing a set of instantiated ports which is overridden by its derived concrete entities. For example, in *Executable* the port list is extracted from the list of *ExecutablePortData*.

3. Workflow Execution

Figure 3. Process execution.

Our process model in combination with the implementation model contains all necessary data for enactment. A couple of workflow engines exist that have already reached the required maturity for production environment (e.g. Kepler [9], MyGrid/Taverna [13]). We do not want to tie our model to a specific workflow system but prefer to have the choice to use the best system for particular requirements.

```
FUNCTION get_port_specs_from_portdata ( port_list : AGGREGATE OF PortData )
        : SET OF EntityName;
    LOCAL
        result : SET OF EntityName;
    END_LOCAL;

    REPEAT i := LOINDEX ( port_list ) TO HIINDEX ( port_list );
        result [ i] := port_list.port_spec;
    END_REPEAT;

    RETURN ( result );
END_FUNCTION;

ENTITY AbstractInstance;
    description : STRING;
    spec : EntityName;
DERIVE
    port_list : SET OF EntityName := [ ];
END_ENTITY;

ENTITY Executable SUBTYPE OF ( AbstractInstance );
    commandline : STRING;
    data_locations : LIST OF ExecutablePortData;
DERIVE
    SELF\AbstractInstance.port_list : SET OF EntityName :=
                        get_port_specs_from_portdata ( data_locations );
END_ENTITY;

ENTITY GuardedTransitionInstance;
    instance : AbstractInstance;
DERIVE
    guardedtransition_spec : EntityName := instance.spec;
WHERE
    has_result :
        SIZEOF ( QUERY (p <* instance.port_list | p = 'result')) = 1;
END_ENTITY;
```

Listing 2. Specification of process implementations in EXPRESS

We keep the model independent of the workflow system by defining transformations that generate the workflow description for the target system from the model. We chose the workflow engine Taverna to demonstrate the approach, because it is intuitive and simple to use, but powerful enough to support sophisticated workflows.

3.1. The Workflow Execution Engine Taverna

A workflow in Taverna [13] consists of inputs, outputs, one or more processors and the data flows between them (see Figure 4). Processors have an interface for inputs and outputs. The outputs of processors can be connected to other inputs or the workflow outputs. The whole workflow is data flow oriented and the order of execution is defined by the data dependencies between processors. A processor is executed as soon as it has got all of its inputs and processors may execute concurrently. The data flow can lead from one output to inputs of more than one processor. If an input is connected to more than one output, the first output to deliver the data "wins". In addition to the data dependencies it is possible to restrict the execution order of processors by defining temporal constraints called "Coordinate from". By defining a "Coordinate from" association between processors A and B, A will only execute when B has completed. The usual method of creating a workflow in Taverna is by using its graphical user interface (GUI). However, all workflows created by using the GUI are passed to the execution engine in the workflow description language Scufl

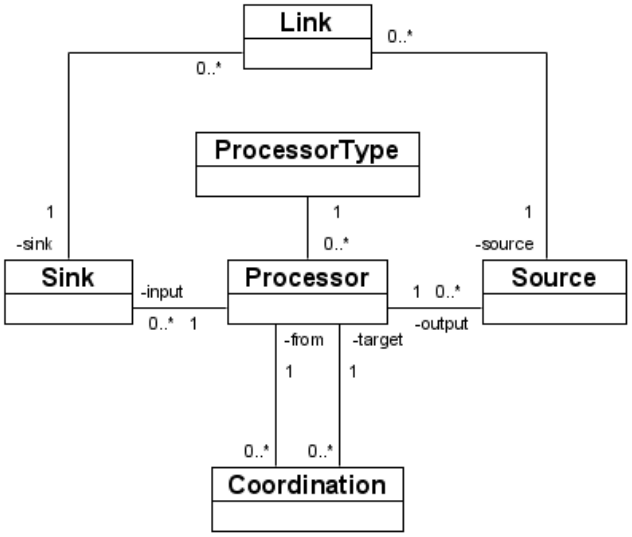


Figure 4. Meta-model of workflows in Taverna.

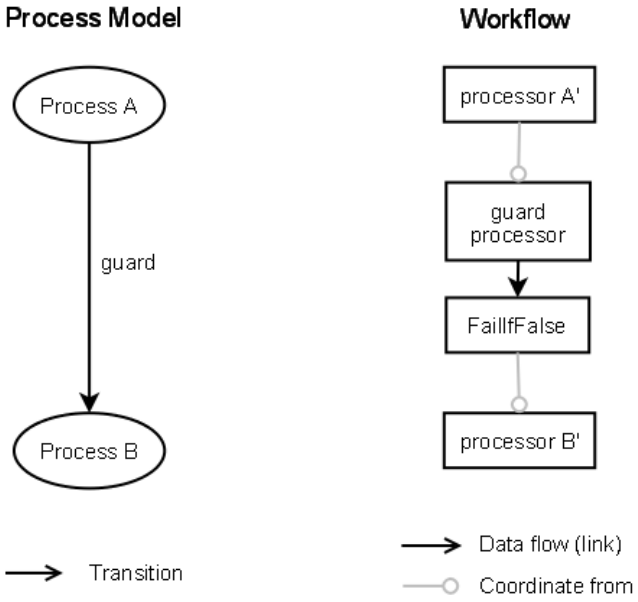


Figure 5. Implementation of a guarded transition in the target workflow.

(Simple Conceptual Unified Flow Language). It is a simple XML-based format representing the elements and links of the workflow and can be used to execute the workflow without the GUI. It contains the workflow description, a couple of processors connected by data flow (link) and control flow edges (coordination) and the inputs (source) and outputs (sink) of the workflow. For our current project, only two types of processors are of interest: the local process FailIfFalse for conditional transitions and the Beanshell Scripting Host for program execution.

3.2. Mapping the Model to a Workflow

The data provided in the process model is sufficient to define the nodes and links in the workflow. The only missing information is the definition of the implementation of the processor nodes. This definition is provided by the implementation model. The four main elements of our process that need to be represented in the workflow description are *Process*, *DataFlow*, *Transition* and *GuardedTransition*. A process maps to a processor, where the interface of the processor is defined by the ports of the process. Data flow objects have a straight-forward equivalent in the workflow description; they are represented by links between processor interfaces. A transition has no direct equivalent in Scufl. The closest representation is a coordinating link, but coordination is more restrictive than a transition. Consider a process A with two transitions coming from process B and C. In our model, the precondition of process A specifies, if the process waits until both or only one of the processes B and C have finished execution. In Scufl, both processes have to finish execution successfully if A is coordinated from B and C. We keep it simple and stick to the behavior of Scufl's coordination element and define that a process has to be reached by all transitions in order to start its execution. The most sophisticated part of the workflow generation is the implementation of a transition with guards, i.e., a GuardedTransition. We implement it in the workflow description by using a processor for the evaluation of the constraint followed by a conditional node (FailIfFalse) and a "coordinate from"-edge to the conditional node (see Figure 5). This rough sketch of how to implement the workflow in Scufl gives a first impression on how to achieve our goal. In order to formalize the transformation we first start by defining the source and target models.

Definition 1
Let $X(P_X, I_X, O_X, D_X, C_X)$ *represent a workflow, where*

P_X ... *set of processors* $\{p_i(i_{i,1} \dots i_{i,M}, o_{i,1} \dots o_{i,N}) | 0 < i \leq N_X\}$, $i_{i,j}$ .. *input* j *of processor* i, $0 < j \leq M_i$, $o_{i,j}$ .. *output* j *of processor* i, $0 < j \leq N_i$,

I_X ... *set of inputs of the workflow, represented as processor outputs* $\{o_{0,j} | 0 < j \leq N_0\}$,

O_X ... *set of outputs of the workflow, represented as processor inputs* $\{i_{0,j} | 0 < j \leq M_0\}$,

D_X ... *data flow between processors represented by a set of links* $\{d_\nu = o_{i,j} \rightarrow i_{k,l}\}$.

C_X ... *set of coordinations* $c(p_i, p_j)$, *coordinate processor* p_i *from* p_j.

Definition 2
Let $M(\Pi_M, R_M, D_M, T_M, G_M)$ denote a process model, where

Π_M *... set of processes* $\{\pi_i | 1 \leq i \leq N_\Pi\}$,

R_M *... set of ports* $\{r_i | 1 \leq i \leq N_R\}$, $R_i \in R_M$ *... set of ports of process* π_i,

D_M *... set of data flow links* $\{l_\nu(r_i \rightarrow r_j)\}$,

T_M *... set of transitions* $\{t_\nu(\pi_i \rightarrow \pi_j)\}$,

G_M *... set of guarded transition* $\{g_\nu(\gamma_\nu, \pi_i \rightarrow \pi_j)\}$, γ_ν *... guard*

Using the two definitions we can write the algorithm to obtain a workflow X from the process model M as follows:

1. $\forall \pi \in \Pi_M$: create processor p_i with inputs I_i and outputs O_i corresponding to the ports in R_π.
2. $\forall t(\pi_i \rightarrow \pi_j) \in T_M$: create "coordinate from" $c(p_i, p_j)$.
3. $\forall g(\gamma, \pi_i \rightarrow \pi_j) \in G_M$:
   (a) create processor p_k with inputs I_γ and output o_γ.
   (b) create processor "FailIfFalse" (p_f) with input i_f.
   (c) create data flow $d_\gamma = o_\gamma \rightarrow i_f$.
   (d) create coordinations $c_{\gamma,1}(p_i, p_k)$ and $c_{\gamma,2}(p_f, p_j)$.
4. $\forall l_\nu(r_i \rightarrow r_j) \in D_M$: create data flow $d_\nu = (o(r_i) \rightarrow i(r_j))$.

4. Example workflow

We tested the workflow generation on a process model for multi-disciplinary design optimization in the automotive industry [15]. Part of the process is the generation of an instance mesh from a geometric model, which is later used by the finite element analysis. The process model of the instance mesh generation contains conditional transitions that select between two possible paths in the process. Depending on the value of a flag (*run_geometry_flag*), either a new mesh is generated or the resulting mesh of a previous run is fetched.

The implementation model contains four executables for the two sub-processes and the evaluation of the constraints on the two guarded transitions, respectively. After applying the transformations presented in the previous section, we get a workflow that can be visualized in the Taverna GUI (see Figure 6) and executed inside the GUI or with the standalone workflow execution engine.

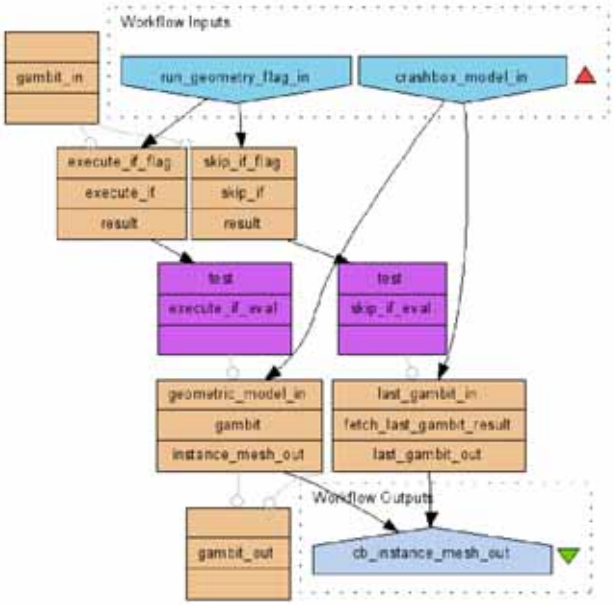


Figure 6. The example workflow in Taverna.

5. Related Work

A number of other process ontologies exist [2, 3, 8]. But to our knowledge, no other work uses EXPRESS to formalize a process model with workflow enactment. An extensive evaluation of other ontologies in the context of industrial design processes can be found in [10].

Our approach consolidates process and artifact ontologies under a common STEP/EXPRESS meta-model. We chose EXPRESS because it comes with huge artifact ontologies and is well-suited for meta-modeling. It can be argued that other standard languages like the *process specification language* PSL are better suited for process modeling. PSL offers a rigorous basis for verifiable semantic definitions, but lacks support for context relationships and needs better definitions of process artifacts [5]. STEP and its application protocols provide in contrast sophisticated domain models for artifact and process modeling.

Mimoune et al. [11] exchange data between heterogeneous database systems using a generic meta-schema formalized in the EXPRESS language to overcome the difficulties arising from different conceptual models for the same implementation and structural differences between implementations of the same conceptual model. Their approach focuses on the definition of mappings between data base schemata.

Work has been done to map from business processes to workflows including ontological mapping between the output of one process and the input of another process. How-

ever, a general mapping from control flow oriented metamodels to data flow oriented systems is hard to achieve [7].

6. Conclusion

Data exchange between different stages of an industrial process is difficult because of the heterogeneity of the involved systems. The common data format standard STEP/EXPRESS helps to overcome the structural differences. The standard defines different encodings and language bindings for its specification language EXPRESS. These "implementation methods" comprise a clear-text and an XML representation and bindings to the programming languages C,C++ and Java. STEP is not intended to provide a common conceptual model, but provides specialized models for domain knowledge. We use process modeling to capture process knowledge explicitly. In addition to allowing easier re-use of process components, explicit process knowledge supports execution tracking so that the provenance of the results is retained.

We propose EXPRESS as the specification language for the ontologies, because thereby we can directly use the domain knowledge specified in the application protocols of the STEP standard. STEP is already used as a common data format for many of the process artifacts in the automotive industry and the data is accessible by our process model without additional structural conversion overhead. Furthermore, writing our models in STEP/EXPRESS allows us to use the same tools as already used for data modeling.

Enactment is another important aspect of process modeling. We have shown that it is possible to use an abstract process model, which is independent of the target platform and build a workflow description for a specific workflow execution engine from this model. We have successfully demonstrated the necessary transformations for the workflow engine Taverna. However, because the process model is independent of the workflow engine, we can use any other workflow execution engine that provides the necessary functionality. In the future, we plan to define transformations for other workflow execution environments and to investigate the integration of domain-specific data models and ontologies.

References

[1] Y. Ait-Ameur, F. Besnard, P. Girard, G. Pierra, and J. C. Potier. Formal specification and metaprogramming in the EXPRESS language. In *Intl.Conf. on Software Engineering and Knowledge Engineering (SEKE)*, pages 181–188, 1995.

[2] B. Chandrasekaran, J. Josephson, and R. Benjamins. The ontology of tasks and methods. In *Proceedings of the 11th Knowledge Acquisition Modeling and Management Workshop, KAW'98*, Banff, Canada, Apr. 1998.

[3] J. Gero and U. Kannengiesser. A function-behavior-structure ontology of processes. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (AI EDAM)*, 21:379–391, 2007.

[4] N. Guarino. Formal ontology and information systems. In N. Guarino, editor, *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems, FOIS'98*, pages 3–15, Trento, Italy, 1998. IOS Press.

[5] A. Gunendran, R. Young, A. Cutting-Decelle, and J. Bourey. Organising manufacturing information for engineering interoperability. In R. Goncalves, J. Muller, K. Mertins, and M. Zelm, editors, *Proceedings of IESA 2007: Enterprise Interoperability II New challenges and Approaches*, pages 587–598. Springer-Verlag London, 2007.

[6] International Organization for Standardization. *ISO 10303-11:1994: Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual*. International Organization for Standardization, Geneva, Switzerland, 1994.

[7] S. Jablonski. On the complementarity of workflow management and business process modeling. *SIGOIS Bull.*, 16(1):33–38, 1995.

[8] Y. Kitamura, Y. Koji, and R. Mizoguchi. An ontological model of device function: industrial deployment and lessons learned. *Applied Ontology*, 1(3–4):237–262, 2006.

[9] B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system, 2005.

[10] F. Maier, W. Mayer, M. Stumptner, and A. Mühlenfeld. Ontology-based process modelling for design optimisation support. In *Third International Conference on Design Computing and Cognition (DCC'08)*, Atlanta, USA, June 2008. To appear.

[11] M. E.-H. Mimoune, G. Pierra, and Y. A. Ameur. An ontology-based approach for exchanging data between heterogeneous database systems. In *ICEIS (1)*, pages 512–524, 2003.

[12] P. H. P. Nguyen and D. Corbett. Building corporate knowledge through ontology integration. *Advances in Knowledge Acquisition and Management*, 4303/2006:223–229, 2006.

[13] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18:1067–1100, 2006.

[14] P. Saliou, A. Plantec, and V. Ribaud. Metaprogramming with EXPRESS and SQL. In *International Workshop Declarative Meta Programming, DMP02. University of Edinburgh*, Dec 2002.

[15] C. Seeling. User manual for the crash-box MDO reference problem. Internal publication, VPAC Ltd, Melbourne, 2007.

[16] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information — a survey of existing approaches. In H. Stuckenschmidt, editor, *IJCAI–01 Workshop: Ontologies and Information Sharing*, pages 108–117, 2001.

Reviewers' Index

A
Alain Abran
Silvia Teresita Acuna
Edward B. Allen
Mikhail Auguston

B
Doo-Hwan Bae
Xiaoying Bai
Maria Teresa Baldassarre
Purushotham Bangalore
Muhammad Ali Barbar
Luciano Barezi
Emese Bari
Saida Benlarbi
Sami Beydeda
Swapan Bhattacharya
Alessandro Bianchi
Jim Bieman
Gary D. Boetticher
Jean-Michel Bruel
Barrett Bryant

C
Kai-Yuan Cai
Danilo Caivano
Gerardo Canfora
Joao W. Cangussu
Giovanni Cantone
Jeffrey C. Carver
Christine W. Chan
Keith C.C. Chan
W.K. Chan
Kuang-Nan Chang
Ned Chapin
I-Chin Chen
Shu-Ching Chen
Yinong Chen
Harry Cheng
Yoonsik Cheon
Peter J. Clarke
Panos Constantopoulos
Kendra Cooper
Maria Francesca Costabile
Karl Cox
Juan J. Cuadrado-Gallego
Alfredo Cuzzocrea

D
Scott Dick
Jin Song Dong
Jing Dong
Philippe Dugerdil
Reiner Dumke
Schahram Dustdar

E
Christof Ebert
Faezeh Ensan
Onyeka Ezenwoye

F
Behrouz Homayoun Far
Martin S. Feather
Robert Feldt
Norman Fenton
Eduardo B. Fernandez
Todd Fitch
Andres Folleco
Jose Fortes
Renata Fortes

G
Kehan Gao
Alessandro Garcia
Felix Garcia
Carlo Ghezzi
Holger Giese
Itana Gimenes

Sebastien Goasguen
Swapna Gokhale
Jeff Gray
Des Greer
Eric Gregoire
Paul Grunbacher

H

Mark Harman
Alan Hartman
Ahmed E. Hassan
Xudong He
Rattikorn Hewett
Mei Hsing
Shihong Huang
Byung-Yeon Hwang

I

Ali Idri
Peter In

J

Clinton Jeffery
Frederic Jouault
Natalia Juristo

K

Audris Kalnins
Sascha Konrad
Gunes Koru
Vinay Kulkarni

L

Mark Last
Jeff Lei
Tao Li
Jennifer Liang
Yingdar Lin
Shih-Hsi Liu
Xiaodong Liu
Yan (Jenny) Liu
Yi Liu
Hakim Lounis
Jian Lu
Zhongyu (Joan) Lu
Heiko Ludwig
Michael R. Lyu

M.

Jose Carlos Maldonado
Antonio Mana
Emilia Mendes
Harald Meyer
Rym Mili
James Miller
Henry Muccini

N

Nachi Nagappan
Martin Neil
Allen Nikora
Elisabetta Di Nitto

O

Mehmet Orgun

P

Manish Parashar
Joontae Park
Witold Pedrycz
Jun Peng
Massimiliano Di Penta
Hoang Pham
Alfonso Pierantonio

R

Rajeev Raje
Sanjay Ranka
Marek Reformat
Robert Reynolds

Daniel Rodriguez
George Roussos
Guenther Ruhe

S

Masoud Sadjadi
Ramon Sagarna
Ahmed Salem
Farshad Samimi
Douglas Schmidt
Naeem Seliya
Tony Shan
Yidong Shen
Martin Shepperd
Simon Shim
Michael Shin
George Spanoudakis
Arndt von Staa
Mark Stamp
Nenad Stankovic
Xiao Su
Rajesh Subramanyan

T

Jeff Tian
Juha-Pekka Tolvanen
Genny Tortora
Mark Trakhtenbrot
Laurence Tratt
Peter Troger
T.H. Tse
Bhekisipho Twala

V

Antonio Vallecillo
Michael VanHilst
Silvia Regina Vergilio
Marlon Vieira

W

Qianxiang Wang
Yingxu Wang
Christiane Gresse von Wangenheim
Tim Weitzel
Laurie Williams
Victor Winter
Eric Wong
Franz Wotawa
Ye Wu

X

Baowen Xu
Zhiwei Xu

Y

Hongji Yang
Huiqun Yu

Z

Cui Zhang
Jing Zhang
Zhi-Hua Zhou
Zhinan Zhou
Hong Zhu
Xingquan Zhu
Eugenio Zimeo
Andrea Zisman

Authors' Index

A

M. Abed, 877
Benjamin Aeschliman, 422
Mohsen Afsharchi, 123
Wasif Afzal, 488
Shir Aharon, 448
Fernanda Alencar, 472
Mauricio Alferez, 779
William Allen, 802
Eduardo Santana de Almeida, 655
Hyggo Almeida, 887
Hyggo O. de Almeida, 599
Zaid Altahat, 905
Alexandre Alvaro, 655
Vasco Amaral, 779
Raquel Anaya, 399
Alain April, 60
Fernando Arango, 399
Joao Araujo, 399, 472, 779
Gonzalo Argote-Garcia, 440
Namfon Assawamekin, 460

B

Linda Badri, 103
Mourad Badri, 103
Rami Bahsoon, 375
Xiaoying Bai, 203, 723
Taiseera Al Balushi, 929
Souvik Barat, 625
Ellen Francine Barbosa, 685, 697
Marcio de O. Barros, 149
Ricardo M. Bastos, 185
Javier Belmonte, 129
Ayse Bener, 143
Tejaswitha Bhavsar, 367
Debmalya Biswas, 531
Keith Bock, 422
Harold Boley, 478
Brandon W. Bonds, 54
Marcos R. S. Borges, 773, 820
Harald Brandl, 393
Jens Bruhn, 48
Olivier Buchwalder, 764
Lofton A. Bullard, 73
Michael Burton, 422

C

Edgar S. Calisaya, 773
Maria Luiza M. Campos, 773
Jaelson Castro, 472
Uiratan Cavalcante, 735
A. P. Cavalcanti, 309
M. Cenk Cavusoglu, 333
Martine Ceberio, 861
Victor Rafael Rezende Celestino, 417
Alma Cemerlic, 791
Christine W. Chan, 857
Shi-Kuo Chang, 4
Yeim-Kuan Chang, 351
Ching-Ming Chao, 603
Ned Chapin, 506
Lorena Chavarria-Baez, 363
Shu-Ling Chen, 33
Tsong Yueh Chen, 16
Zhenyu Chen, 494
Harry H. Cheng, 565
Yoonsik Cheon, 861
Radhika Chhabra, 715
Chih-Ping Chu, 351
Peter J. Clarke, 440, 500
Cecilia Claudio, 1
Nelly Condori-Fernandez, 22
Oscar Corcho, 929
Henk Corporaal, 785
Chessman Correa, 67
Antonio Cortes, 861
Evandro Costa, 599
R. Costa, 309
Arnaud Counet, 60
Antonio Marcio Ferreira Crespo, 417
Jesus Cristobal, 667
Xiaofeng Cui, 321

D

Jian Dai, 223, 561
Yu Dai, 215, 853
Gargi Dasgupta, 814
Jose Maria N. David, 820
Debzani Deb, 808
Yi Deng, 3
Nima Dezhkam, 26
Oscar Dieste, 769
Oguz Dikenelli, 741
Qin Ding, 381
Yulin Ding, 752
Donna Djordjevich, 565
Jing Dong, 454
Dennis J. Drown, 279
Lucas Drumond, 638
Gengshen Du, 137
Edward B. Duffy, 303
Philippe Dugerdil, 129
Reiner R. Dumke, 918

E

Nina Edelweiss, 525
Raimund Ege, 411
Tzilla Elrad, 581, 905
Maria Claudia F. P. Emer, 357
Wolfgang Emmerich, 375
E. Zeynep Erson, 333
Onyeka Ezenwoye, 649, 814

F

Li Fan, 613
Behrouz H. Far, 123, 466
Pedro Porfirio Muniz Farias, 847
Ali Fatolahi, 619
Robert Feldt, 488
Chenhua Feng, 155
Paula Fernandes, 758
Alfredo Fernandez-Valmayor, 667
Adriana M. C. M. Figueiredo, 173
Olympio C. Silva Filho, 887
Paulo N. Cruz Filho, 197
Todd Fitch, 631
Andres A. Folleco, 73
Liana Fong, 814
Lisandra M. Fontoura, 179
Richard Ford, 802
Renata Pontin de Mattos Fortes, 655
Gordon Fraser, 393, 709
Yujian Fu, 440
M. Muztaba Fuad, 315
Mathias Funk, 785
F. Furtado, 309

G

Suyog Gaidhani, 555
Renata de Matos Galante, 525
Irbis Gallegos, 273
Jerry Gao, 631, 715
Kening Gao, 853
Luis Garcia, 261
Rogerio Eduardo Garcia, 685
Vinicius Cardoso Garcia, 655
Ann Gates, 273
Tom Gelhausen, 691
Blaise Genest, 531
C. Ghezzi, 255
Sudipto Ghosh, 873
Tomas San Feliu Gilabert, 42
Rosario Girardi, 638, 735
Olivier Le Goaer, 387
Seyed Koosha Golmohammadi, 643
Jin-gang Guo, 38

H

Laura Haas, 2
Naji Habra, 60
Jason O. Hallstrom, 303
Klaus Marius Hansen, 345, 893
Lasse Harjumaa, 91
Keqing He, 830
Xiao-yang He, 38
Xudong He, 440

Peter Henderson, 327
Yanelis Hernandez, 500
S. Herr, 339
Rattikorn Hewett, 703
Jonathan Hittle, 873
Erika Nina Hohn, 685
Jason Honda, 565
Seongsoo Hong, 573
Lei Hu, 842
Lifeng Hu, 867
Songlin Hu, 209
Wen Shen Huang, 381
Yu-Chun Huang, 33
Sebastian Hudert, 587
Oliver Hummel, 232
Byung-Yeon Hwang, 219

I

Colin J. Ihrig, 4
Magda G. Ilieva, 478
Jin Woo Im, 219

J

Vijayananda Jagannatha, 555
Andrea Janes, 191
Dietmar Jannach, 405
Wenpin Jiao, 250
Ying Jin, 367
Mario Jino, 357

K

Gail Kaiser, 867
Selim Kalayci, 814
Taghi M. Khoshgoftaar, 73, 279, 519
Amir A. Khwaja, 97
Phongphun Kijsanayothin, 703
Beomjin Kim, 422
Tariq M. King, 500
Joseph M. Kizza, 791
Christos Kloukinas, 117
C. Kolski, 877
Sven J. Korner, 691
Nicholas A. Kraft, 54, 85
Uira Kulesza, 745, 779
Vinay Kulkarni, 625
Martin Kunz, 918
Karen Kwok, 631
Gihwon Kwon, 537, 543

L

Jouni Lappalainen, 91
K. Laufer, 339
Jaqueline I. Lavandera, 428
Gary T. Leavens, 861
Bum-Suk Lee, 219
Adriana Leite, 735
Hanna Leskinen, 91
Timothy C. Lethbridge, 619
Huaizhang Li, 561
Juanzi Li, 203
Mengjun Li, 795
Mingshu Li, 561
Weigang Li, 417
Xiaoou Li, 363
Yang Li, 238
Zhoujun Li, 795
Wrihang Roberto Liang, 715
Ying Liang, 209
Huimin Lin, 16
Bin Liu, 830
Chien-Hung Liu, 33
Dapeng Liu, 161
Feng Liu, 795
Guoliang Liu, 238
Jiakun Liu, 573
Jing Liu, 830
Tao Liu, 723
Marta Lopez, 769
Pericles Loucopoulos, 929
Emerson Loureiro, 599, 887
Joel Pinho Lucas, 607
Carlos J. P. de Lucena, 745
Daniel Lucredio, 655

Heiko Ludwig, 587

M

Khaled Mahbub, 117
Mark Mahoney, 581
Franz Maier, 935
Jose Carlos Maldonado, 685, 697
Brian A. Malloy, 303
Paolo Maresca, 4
Gerald Marin, 802
Tiago Cordeiro Marques, 847
Santiago Matalonga, 42
Wolfgang Mayer, 935
Philip K. McKinley, 881
D.A. Meedeniya, 371
Hong Mei, 321
S. R. L. Meira, 309
Silvio Romero de Lemos Meira, 655
Steffen Mencke, 918
Jorge Merino, 667
Robert Merkel, 16
Marcio Gurjao Mesquita, 847
Boleslaw Mikolajczak, 267
Ali Mili, 448
Sameer Mohammed, 291
Heidi Moisanen, 91
Alberto L. Moran, 428
Sandro Morasca, 297
Ana Moreira, 399, 472, 779
María N. Moreno, 607
V. Moura, 309
Abdolmajid Mousavi, 466
Arndt Muhlenfeld, 935
Jurgen Munch, 167
Christian Murphy, 867
Leonardo Murta, 67, 758

N

Chaitanya Nadkarni, 448
Elisa Y. Nakagawa, 697
Haruka Nakaoa, 167
Antonio Navarro, 667
Roman Neruda, 569
Julio Cesar Campos Neto, 847
Mihai Nica, 899
Changhai Nie, 484
Camila Nunes, 745
Ingrid Nunes, 745

O

Omar Ochoa, 273
Kleinner Oliveira, 912
Toacy Oliveira, 912
Toacy C. Oliveira, 185
Flavio Oquendo, 244
Michael J. Oudshoorn, 808
Mourad-Chabane Oussalah, 387

P

Fernando Paniagua, 577
Jiyong Park, 573
Sachoun Park, 537, 543
Tauhida Parveen, 111, 802
Oscar Pastor, 22
Mitul Patel, 929
Dennis S. Patrone, 227
Jairo Pava, 500
John Paxton, 808
Witold Pedrycz, 643
Bernhard Peischl, 899
Tu Peng, 454
Eliana B. Pereira, 185
Marcos F. Pereira, 599
A.S.Perera, 371
Angelo Perkusich, 599, 887
Claude Petitpierre, 764
Jose A. Pino, 820
Charnyote Pluempitiwiriyawej, 460
Tytti Pokka, 91

Roberto Tom Price, 179
Piet van der Putten, 785

Q

Zawar Qayyum, 244
Bo Qu, 484

R

Damith C. Rajapakse, 923
Bina Ramamurthy, 227
Felicidad Ramos, 769
Bonnie Ray, 155
Abhinay Reddyreddy, 512
Marek Reformat, 643
Marcio de M. Ribeiro, 599
Ana C. Riekstin, 697
Steve Roach, 261, 273
Oscar M. Rodriguez-Elias, 428
Guenther Ruhe, 137
Vasile Rus, 291

S

Deise de Brum Saccol, 525
S. Masoud Sadjadi, 649, 814
Salamah Salamah, 261, 273
Clenio F. Salviano, 173
Farshad A. Samimi, 881
Pedro R. Falcone Sampaio, 929
Sherri M. Sanders, 824
Danilo F. S. Santos, 887
Joao Santos, 779
Kamran Sartipi, 26, 842
Gregor Scheithauer, 12, 549
Andreas Schönberger, 593
Stefan Seedorf, 232
Saddys Segrera, 607
Naeem Seliya, 79, 279
Abdelhak-Djamel Seriai, 387
J. Shafaee, 339
Alan B. Shaffer, 673
Leyuan Shi, 440
Michael E. Shin, 577
Sajjan Shiva, 291
Alberto Sillitti, 191
Carla Silva, 472
Fabio Silva, 638
Nishadi De Silva, 327
Jukka Sirvio, 91
Harvey Siy, 613
John C. Sloan, 519
Darunee Smavatkul, 703
Randy K. Smith, 54
Goncalo Soares, 345
Stephane S. Some, 619
Hui Song, 250
Yicheng Song, 209
M. Soui, 877
Andre Sousa, 779
Bueno Borges de Souza, 417
George Spanoudakis, 117, 661
Mark Stamp, 223
Nenad Stankovic, 434
Rod Strong, 422
Markus Stumptner, 935
Daniel St-Yves, 103
Giancarlo Succi, 191
Yanchun Sun, 250, 321
Thanwadee Sunetnanta, 460
Ramyashree Swamyo, 715

T

Marta S. Tabares, 399
Luca Vetti Tagliati, 679
Luay Tahat, 905
G. Tamburrelli, 255
Dalila Tamzalit, 387
Fatih Tekbacak, 741
Yu-xin Teng, 38
Valentina Ternelli, 4
G. K. Thiruvathukal, 339
Jiuming Tian, 209
Scott Tilley, 111, 802
Andy Tinkham, 111

Candemir Toklu, 549
Richard Torkar, 488
Adam Trendowicz, 167
Chi K. Tse, 830
Theocharis Tsigritis, 661
Tugkan Tuglular, 741
Burak Turhan, 143

U
Joseph E. Urban, 97
G. Uster, 877

V
Corina Vela, 273
Silvia Regina Vergilio, 197, 357
Balaji Viswanathan, 814
Aurora Vizcaino, 428
Thomas Vogel, 48

W
Daoming Wang, 16
Qing Wang, 161, 561
Xinghua Wang, 250
Ya-sha Wang, 38
Kevin S. Webb, 85
Jun Wei, 238
Martin Weiglhofer, 709
Claudia Werner, 67, 758
Guido Wirtz, 12, 48, 339, 549, 587, 593
Franz Wotawa, 393, 709, 836, 899
Leon Wu, 867
Weibiao Wu, 79
Yan Wu, 613
Yuxiang Wu, 857

X
Xiaolin Xi, 573
Ma Xiang, 715
Junchao Xiao, 161, 561
Sai Xiao, 321
Lizi Xie, 161
Baowen Xu, 484, 494
Haiping Xu, 512
Zhiwei Xu, 79

Y
Chi-Lu Yang, 351
Lei Yang, 215, 853
Li Yang, 411, 791
Zilan (Nancy) Yang, 123
Xinyu You, 203

Z
Bin Zhang, 215, 853
Cui Zhang, 824
Du Zhang, 219, 285
Weishan Zhang, 345, 893
Xiaofang Zhang, 484, 494
Yan Zhang, 752
Zhoulan Zhang, 4
Yajing Zhao, 454
Cheng Zhong, 123
Bin Zhou, 729
Ti Zhou, 795
Xin Zhou, 155
Hong Zhu, 729
Thomas Zimmermann, 137

SEKE 2009 Call For Papers

The Twenty-First International Conference on Software Engineering and Knowledge Engineering

Hyatt Harborside at Logan Int'l Airport, Boston, USA
July 1 - July 3, 2009

Organized by
Knowledge Systems Institute Graduate School

The Twenty-First International Conference on Software Engineering and Knowledge Engineering (SEKE'09) will be held at the Hyatt Harborside at Boston's Logan Int'l Airport, Boston, USA, July 1-3, 2009.

The conference aims at bringing together experts in software engineering and knowledge engineering to discuss on relevant results in either software engineering or knowledge engineering or both. Special emphasis will be put on the transference of methods between both domains.

TOPICS

Solicited topics include, but are not limited to:
Agent architectures, ontologies, languages and protocols
Agent-based learning and knowledge discovery
Agent-based software engineering
Autonomic computing
Agent-based auctions and marketplaces
Adaptive Systems
Artificial Intelligence Approaches to Software Engineering
Artificial life and societies
Automated Reasoning
Automated Software Design and Synthesis
Automated Software Specification
Component-Based Software Engineering
Computer-Supported Cooperative Work
Data cleansing and noise reduction
Data streams and incremental mining
Data visualization
E-Commerce Solutions and Applications
Embedded and Ubiquitous Software Engineering
Electronic Commerce
Enterprise Software, Middleware, and Tools
Formal Methods
Human-Computer Interaction
Industry System Experience and Report
Integrity, Security, and Fault Tolerance
Interface agents
Knowledge Acquisition
Knowledge-Based and Expert Systems
Knowledge Representation and Retrieval
Knowledge Engineering Tools and Techniques
Knowledge Visualization
Learning Software Organization
Measurement and Empirical Software Engineering
Middleware for service based systems
Mobile agents
Mobile Commerce Technology and Application Systems
Mobile Systems
Multi-agent systems
Multimedia Applications, Frameworks, and Systems
Multimedia and Hypermedia Software Engineering
Ontologies and Methodologies
Patterns and Frameworks
Pervasive Computing
Process and Workflow Management
Programming Languages and Software Engineering
Program Understanding
Quality of services
Reflection and Metadata Approaches
Reliability
Requirements Engineering
Reverse Engineering
Runtime service management
Secure mobile and multi-agent systems
Semantic web
Service-centric software engineering
Service oriented requirements engineering
Service oriented architectures
Service discovery and composition
Service level agreements (drafting, negotiation, monitoring and management)
Smart Spaces
Soft Computing
Software Architecture
Software Assurance
Software Domain Modeling and Meta-Modeling
Software dependability
Software economics
Software Engineering Case Study and Experience Reports
Software Engineering Decision Support
Software Engineering Tools and Environments
Software Maintenance and Evolution
Software Process Modeling
Software product lines
Software Quality
Software Reuse
Software Safety
Software Security
Swarm intelligence
System Applications and Experience
Time and Knowledge Management Tools
Tutoring, Documentation Systems
Uncertainty Knowledge Management
Validation and Verification
Web and text mining
Web-Based Tools, Applications and Environment
Web-Based Knowledge Management
Web-Based Tools, Systems, and Environments
Web and Data Mining

CONFERENCE SITE (HOTEL INFORMATION)

The SEKE 2009 Conference will be held at the Hyatt Harborside at Boston's Logan Int'l Airport, Boston, USA. The hotel has made available for these limited dates (6/30 - 7/4/2009) to SEKE 2009 attendees a discount rate of $149 US dollars for single/double, not including sales tax.

INFORMATION FOR AUTHORS

Papers must be written in English. An electronic version (Postscript, PDF, or MS Word format) of the full paper should be submitted using the following URL: http://conf.ksi.edu/seke09/submit/SubmitPaper.php. Please use Internet Explorer as the browser. Manuscript must include a 200-word abstract and no more than 6 pages of IEEE double column text (include figures and references). Workshop papers should be submitted to the workshops directly.

INFORMATION FOR REVIEWERS

Papers submitted to SEKE'09 will be reviewed electronically. The users (webmaster, program chair, reviewers...) can login using the following URL: http://conf.ksi.edu/seke09/review/pass.php.

If you have any questions or run into problems, please send e-mail to: seke@ksi.edu.

SEKE 2009 Conference Secretariat
Knowledge Systems Institute Graduate School
3420 Main Street
Skokie, IL 60076 USA
Tel: 847-679-3135
Fax: 847-679-3166
E-mail: seke@ksi.edu

IMPORTANT DATES

March 1, 2009 — *Paper submission due*
April 1, 2009 — *Notification of acceptance*
May 1, 2009 — *Camera-Ready Copy*