Savitri Bevinakoppa,
Luís Ferreira Pires and
Slimane Hammoudi (Eds.)

# Web Services and Model-Driven Enterprise Information Services

**Proceedings of the Joint Workshop on Web Services and Model-Driven Enterprise Information Services, WSMDEIS 2005**
In conjunction with ICEIS 2005
Miami, U.S.A., May 2005

**INSTICC PRESS**
*Portugal*

Volume Editors

Savitri Bevinakoppa
Royal Melbourne Institute of Technology,
Australia

Luís Ferreira Pires
CTIT, University of Twente,
The Netherlands

and

Slimane Hammoudi
ESEO, Angers ,
France

Proceedings of the Joint Workshop on
Web Services and Model-Driven
Enterprise Information Services
WSMDEIS 2005
Miami, U.S.A., May 2005.
Savitri Bevinakoppa
Luís Ferreira Pires
and Slimane Hammoudi (Eds.)



Printed in Portugal

ISBN 972-8865-27-9
Depósito Legal: 224491/05

# Foreword

*Web services* and *Model-driven development* are two emerging research fields and have been receiving a lot of attention in the recent years. New approaches on these two areas can bring many benefits to the development of information systems, distribution flexibility, interoperability, maintainability and portability. Nevertheless, these emerging fields pose new promising challenges to the research community. Some of the current challenges in the web services field are service composition, support for quality of service, security and integration of legacy systems; in the model-driven development field are the mappings between metamodels, model transformations, semantic distance and traceability.

This volume contains the *Proceedings of the 3rd International Workshop on Web Services: Modeling, Architecture and Infrastructure* (WSMAI 2005), and *the Proceedings of the First International Workshop on Model-Driven Enterprise Information Systems* (MDEIS 2005).

WSMAI 2005 and MDEIS 2005 aim at serving as a forum for researchers and practitioners to meet and share expertise in the fields of Web services and Model-driven development, respectively. WSMAI 2005 has received fifteen papers; nine papers were accepted for regular oral presentation and three papers were accepted as posters. MDEIS 2005 has received ten papers; five papers were accepted for regular oral presentation and three papers were accepted as posters.

We would like to take this opportunity to thank the people who have contributed to WSMAI 2005 and MDEIS 2005. We would like to thank the members of the WSMAI 2005 and MDEIS 2005 Program Committees for the terrific job they did in evaluating papers and the authors for their paper contributions in shaping the final programs. Finally, special thanks to Joachim Filipe and Vitor Pedrosa for their hard work in making the workshops and this volume possible.

We wish you an exciting, fruitful workshop, and an unforgettable stay in the lovely city of Miami.

Workshop Chairs – WSMDEIS 2005

Savitri Bevinakoppa
Royal Melbourne Institute of Technology,
Australia

Luís Ferreira Pires
CTIT, University of Twente,
The Netherlands

Slimane Hammoudi
ESEO, Angers ,
France

## Workshop Chairs

Savitri Bevinakoppa
Royal Melbourne Institute of Technology,
Australia

Luís Ferreira Pires
CTIT, University of Twente,
The Netherlands

and

Slimane Hammoudi
ESEO, Angers ,
France

## Program Committee

Jen-Yao Chung, (IBM, USA)
Alex Delis, (Polytec University, NY, USA)
Jiankun Hu (RMIT University, Australia)
Steve Vinoski (IONA, USA)
Albert Y. Zomaya, (CISCO, USYD , Australia)
MarianoBelaunde (France Telecom, France)
Bernard Coulette (GRIMM, Université de Toulouse, France)
Philippe Desfray (Softteam, France)
Marlon Dumas (QUT University, Australia)
Anastasius Gavras (Eurescom, Germany)
Sune Jacobson (Telenor, Norway)
Santosh Kumaran (IBM, USA)
Jean Louis Sourrouille (INSA , Université de Lyon, FRANCE)
Andreas Tolk(VMASC, USA)
Antonio Vallecillo (ESTI, Universidad de Malaga, SPAIN)
Marten van Sinderen (University of Twente, Netherland)

# Table of Contents



## Full Papers

## Posters

# Papers

# An XML-Based System for Configuration Management of Telecommunications Networks Using Web-Services

Adnan Umar, James J. Sluss Jr. and Pramode K. Verma

The University of Oklahoma,
4502 E. 41st Street, Building 4, Room 4403
Tulsa, Oklahoma 74135, USA
{umar,sluss,pverma)@ou.edu

**Abstract.** As the utilization and the application base of the Internet grows, the need for an improved network management system becomes increasing apparent. It is generally accepted that SNMP is not capable of tackling the arising network management requirements and needs to be replaced. Also, configuration management has been identified as one of the most desired network management functionality. Recent research publications suggest a growing interest in replacing SNMP by a Web Services (XML)-based network management solution. In this paper we present our methodology and design of our complete XML-based network management system developed with the specific aim of performing configuration management. [1], [2]

## 1 Introduction

As the utilization and the application base of the internet grows, the need for an improved management system becomes increasingly apparent. The management of the Internet is traditionally based on the framework of SNMP. The SNMP was designed almost fifteen years ago to address the network management needs of that time. Back then, the networking environment was very different. The primary goals of SNMP were to perform device-level management, be extensible, and efficient in using communication and processing resources. Today, advances in technology have dramatically changed the networking environment. This dramatic change has altered the management requirements significantly. Scarcity of bandwidth and processing power is no longer an issue and heterogeneous networks are commonplace. Configuration management has been identified as the most desired management functionality. Inadequacies of SNMP and the need for a new management technology have also been brought to light. The two prime candidate technologies for the development of a new management system appear to be XML and Java. Currently many companies and standards bodies are working on developing an XML-based network management system. To study the use of XML in network management system within the wider research community, there needs to be a design and open-source implementation that would facilitate research. In this paper, we present our methodology and design of our complete XML based network management system developed with the specific aim studying configuration management. To the best of our knowledge, no such effort is being undertaken at this time [1], [2], [3], [4].

## 2 Methodology

The task of developing an XML based system includes specifying design requirements, choosing the appropriate XML technologies, and testing the XML design using generic software tools. This task can become very challenging since the XML technologies are constantly changing and the software tools are often playing catch-up. For this reason we have adopted a methodology with which we can organize our design process. Our methodology consists of three activities. An 'activity' can be defined as the process of taking iterative-steps to accomplish a task. First, we decompose our management problem into functions and map these functions to XML technologies. Second, using the results of our first activity, we piece together our XML design. Third, we test our design by implementing a subset of it using generic tools.

## 3 Network Management Functions

Our first activity in designing the management system was to identify the major functions and map them to the appropriate XML technology. We have identified the following functions that we believe an XML based network management system using Web Services requires. This approach allows us to keep track of the evolving XML technologies and facilitates the implementation process. The guideline we followed for mapping the network management functions and XML technology is that, they should be closely aligned such that only generic tools are required for implementation. That is to say, if implemented correctly using generic XML tools, our design should perform the required configuration management task. These network management functions and their respective XML technologies are listed below:

- **Defining structure of management information (XML Schema [9], [10], [11]).** The ability of representing a very large variety of information in a homogeneous fashion is crucial to success of a management system. This function can be performed by the XML Schema technology. XML Schema is an XML language that is capable of defining the structure of a XML document. That is, it specifies which tags are permitted and in which nesting order, and constrains on the number of occurrences of a particular tag, etc. A tag can be made optional or required and the value's data-type can be declared. Another impressive feature of XML Schema is its ability to validate a XML document. This feature can be used to reduce code complexity by catching erroneous XML documents, that do not match the defined tag-value structure, before they get passed on to the application.
- **Handling the management data (XML Document [6]).** All information needs to be represented in a form that allows information to be accessed, modified, searched, and retrieved. This function can be performed by XML documents. These documents will be based on a XML Schema that defines its structure and the way information should be represented and

can be validated against it. XML documents are also a convenient way for storing information.

- **Navigating in the management documents (XPath [7] and XPointer [18]).** In an XML based management system, all information exists in the form of XML documents. The number, size and complexity of these documents can get rather large. Therefore, a function is required that is capable of navigating through a maze of XML documents. The XML community has addressed this issue by developing two specification called XPath and XPointer. XPath is a recommendation that defines how nodes within an XML documents can be accessed by forming an expression. These expressions play a major role in other XML standards, such as, XSLT and XQuery. XPointer is built on XPath and includes URI addressing making it possible to address fragments of an XML file.
- **Providing an interface between XML document and management applications (DOM [12] and SAX APIs).** In any XML based system there is a need for an interface between XML documents and the application. Although applications can treat the XML documents as a text document and use their own parsing scheme, it is much better to use a standard parser. Currently there are two popular XML interfacing standard parsers: SAX and DOM.

  SAX (Simple API for XML) is a joint development of the members of the XML-DEV mailing list. SAX is very simple, easy to learn, and not demanding on resource such as memory. Various SAX parsers are available for Java, C++, Python, Perl and Delphi.

  DOM (Document Object Model) is a complete interface to an XML document. Using DOM applications can parse, retrieve, add, modify, and delete sections of the XML documents. Since DOM stores the entire XML document in memory, it can be very intensive on resources. Popular implementations for DOM are MSXML from Microsoft for Windows and Apache Soft ware Foundation's Xerces that exists in Java and C++.
- **Changing format of XML documents (XSLT [8]).** An important issue for XML in general is document transformation. Often information stored in XML documents is formatted for the purpose of storage. To make the information useful, often re-formatting is required. For his purpose XML community has developed XSLT. XSLT is a powerful technology capable of transforming one document into another document with a different format. Several implementation tools for XSLT are available for various platforms, such as, Java, C++, and Perl.
- **Describing the management interface (WSDL [13]).** Since we are employing Web Services we cannot avoid the use of WSDL (Web Services Description Language). WSDL is an XML language that describes a web service. WSDL compilers are used to generate executable code and are available for various platforms and programming languages.

  SNMP systems handle this function by providing MIBs can be used by the MIB compilers. Data can by accessed by SNMP functions such as Get, GetNext, and Set.

- **Transporting parts of or entire XML documents (SOAP [14], [15], [16]).** This function is responsible for providing the means for communication. We require this means of communication to be capable of using any transportation protocol. We can map this function to the XML standard called SOAP. SOAP is one of the Web Services standards developed for exchange of information. SOAP is not tied to any transport protocol and can use SMTP, FTP, etc. However, HTTP is the most popular transportation protocol used by SOAP.

## 4 Design

The second activity in our methodology is creating a design. This design is realized by using the results of the first activity. The proposed design (Fig. 1) creates a system of XML documents that are capable of representing all desired management functions. These documents can be implemented on any OS using any of the large number of software available for XML. Currently, XML programming tools are available for all major software platforms including .NET, COM, C/C++, and Java. Furthermore, XML can also be implemented using scripting languages such as, PHP, Cold Fusion, Perl, Python and Tcl. For every design there needs to be a set of requirements that are to be fulfilled. We are using the list of requirement mentioned in [1]. Following is the list of requirement and an explanation on how our design fulfills them:

- **Maintaining a clear distinction between configuration and operational data.** In our design we maintain a clear distinction between configuration and operational data by keeping them in separate XML documents. Both can have their respective XML Schema document to help validate the XML documents containing the data. It also appears desirable to handle the two separately in management application as well.
- **Providing functionality to download and upload a small part or entire configuration files.** Our design is capable of transporting XML documents by using SOAP between the network manager and network device. These XML documents can be a small part of or entire configuration file. Operational information is also to be passed using this mechanism.
- **Ensuring that configuration data is kept in text format for interoperability.** Everything in XML is kept in a text format. Therefore, this requirement is met inherently by using the XML technology.
- **Enabling the devices to hold multiple configurations, one of which can be active at any given time.** If one configuration can be stored in one XML document, then by having multiple XML files multiple configurations can be kept in one device. To have only one active however, appropriate functions need to be implemented at the application level.

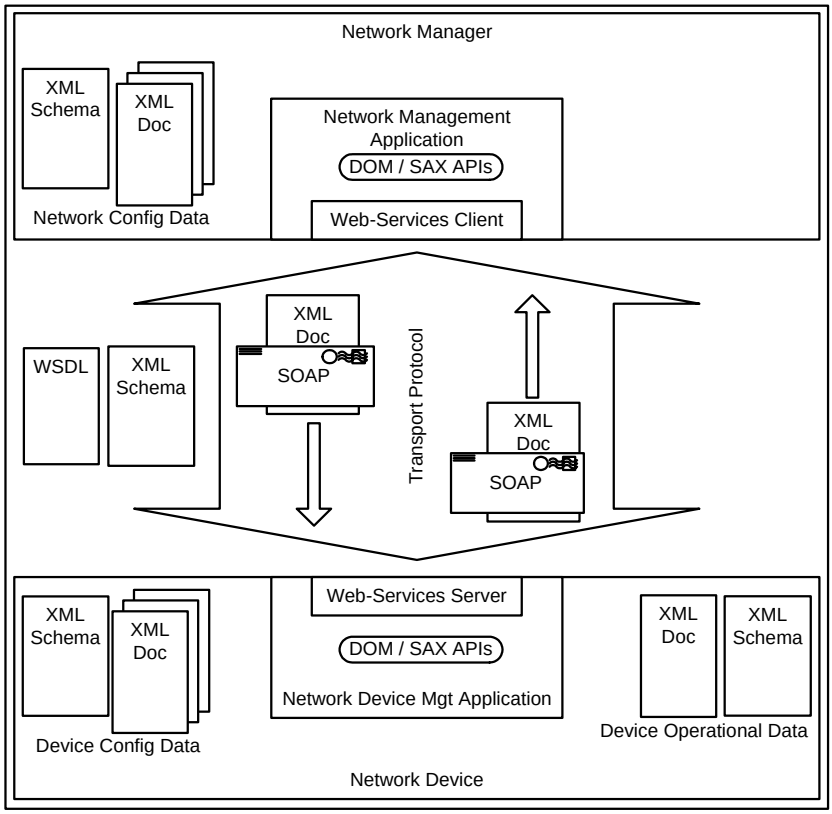



**Fig. 1.** Proposed XML-Based system for configuration management using web-services

- **Providing a simplified mechanism for coordinated activation of configuration taking into account loss of connectivity during a management transaction.** A simplified mechanism for coordinating activation of configuration will need to be handled at the application level. In our design SOAP provides the connectivity and the application can benefit from its functionality.
- **Being easy to use and cost effective. Ease of use and cost effective requirement is met inherently by using XML.** Unlike SNMP, XML is a generic technology which makes XML tools, open-source software, and developers with XML expertise much easier to find and is cost effective.

## 5 Testing

Testing of the XML design is the third activity in our methodology. In this section we show how we tested our XML design. The main objective of this activity is to verify that our design can be implemented using generic tools. We did this verification by implementing a subset of our design that included all the functions that we identified

in our first activity. In our implementation we are using Jakarta Apache Tomcat [20] and Apache Axis [19] as the web-server and SOAP engine respectively. The challenge we face is that Apache Axis's does not fully support all the features of web services and supported features are often difficult to implement. For these reasons we follow the technique suggest in [5] (Fig. 2). By adopting this technique we can easily achieve document/literal style SOAP encoding and XML Schema validation support by only performing a few steps. This technique demonstrates how Castor [22] can be used in conjunction with Apache Axis to easily implement any web-service. We tested our network management web-service using XML Spy [23] by placing a sample XML document with sample configuration data. JDOM [21] was used to access the XML document. The steps of the implementation can be found in [5].

Some problems however, still remain. Notification operation is not supported by Apache Axis's WSDL2Java tool. Furthermore, it is not entirely clear how web-services handle the notification operation, since there are no clear guidelines on how a client can register itself with the server. A custom solution can be developed, but will lead to interoperability issues. We believe that these issues will be resolved by the XML community in the near future.

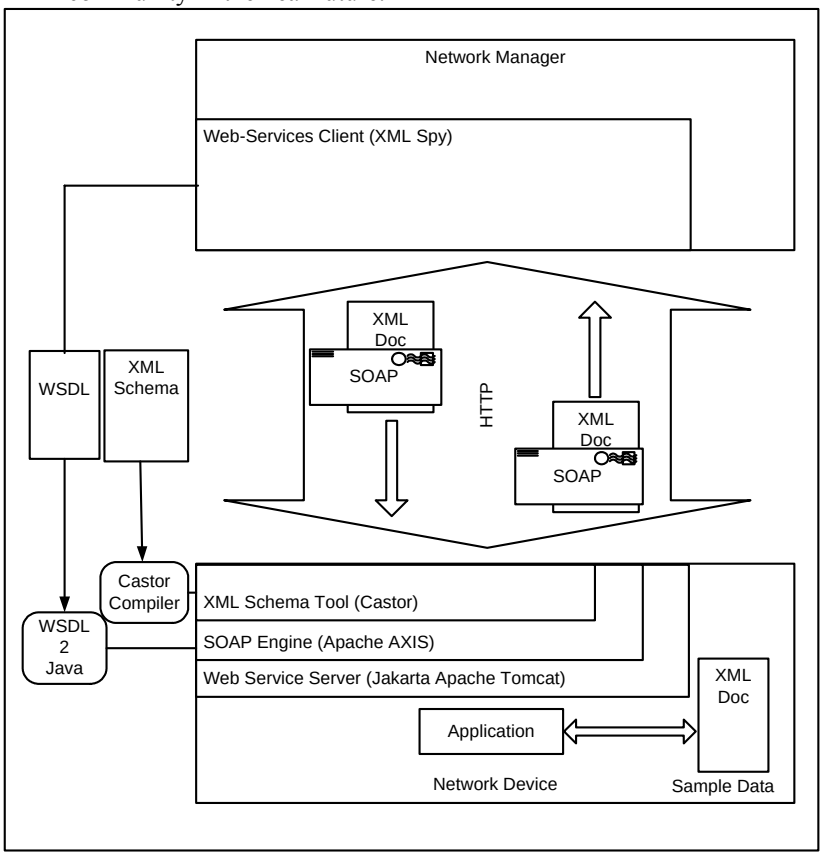



**Fig. 2.** Prototype and testing setup

## 6 Conclusion and future work

In this paper we have presented our methodology and design of our XML-based network management system using web services. This methodology and design is intended to provide a framework which will enable us to study configuration management problems using XML based management system in great detail. Future work currently underway entails an open source implementation of our design. This implementation is intended to be a tool for research related to configuration management.

## Acknowledgements

Authors would like to acknowledge Keith Allen, Weijing Chen, and Will Chorley of SBC Labs for providing an opportunity to work with the XML technology in network management.

## References

1. Jürgen Schönwälder, Akio Pras, Jean-Philippe Martin-Flatin, "On the Future of Internet Management Technologies," IEEE Communication Magazine, October 2003
2. L. Sanchez, K. McCloghrio, and J. Saperia, "Requirements for Configuration Management of IP-based Networks," RFC 3139, June 2001
3. Jae-Oh Lee, "Enabling Network Management Using Java Technologies," IEEE Communication Magazine, January 2000
4. Frank Strauβ, Torsten Klie, "Towards XML Oriented Integrated Network Management," 2003. IFIP/IEEE Eighth International Symposium, March 2003
5. Kevin Gibbs, Brian D Goodman, and Elias Torres, "Create Web Services using Apache Axis and Castor," IBM developersWorks, http://www-136.ibm.com/developerworks/
6. T. Bary, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, October 2000
7. J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, Inso Corp. and Brown University, November 1999
8. J. Clark. XSL Transformation (XSLT) Version 1.0. W3C Recommendation, November 1999
9. David C. Fallside. XML Schema Part 0: Primer. W3C Recommendation, IBM, May 2001
10. Henry. S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn, XML Schema Part 1: Structures, W3C Recommendation. University of Edinburgh, Oracle Corporation, Commerce One, Lotus Development Corporation. May 2001
11. Paul V. Biron and Ashok Malhotra. XML Schema Part 1: Datatypes. Kaiser Permanente, Microsoft. May 2001
12. L. Wood, et al. Document Object Model (DOM) Level 1 Specification. W3C Recommendation, SoftQuad, October 1998
13. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, Microsoft, IBM, March 2001
14. Nilo Mitra. SOAP Version 1.2 Part 0 : Primer. W3C Recommendation, Ericsson, June 2003

15. Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik F. Nielsen. SOAP Version 1.2 Part 1: Messaging Framework. Microsoft, Sun Microsystems, IBM, Canon. W3C Recommendation. June 2003
16. Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik F. Nielsen. SOAP Version 1.2 Part 2: Adjuncts. Microsoft, Sun Microsystems, IBM, Canon. W3C Recommendation. June 2003
17. Hugo Haas, Oisin Hurley, Anish Karmarkar, Jeff Mischkinsky, Mark Jones, Lynne Thompson, Richard Martin. SOAP Version 1.2 Specification Assertion and Test Collection. W3C, IONA Technologies, Oracle Corp. AT&T, Unisys, Active Data Exchange.W3C Recommendation June 2003
18. Steve DeRose, Ron Daniel Jr, Eve Maler, Jonathan Marsh, Norman Walsh. XML Pointer Language (XPointer). Brown University, Interwoven, Arbortext Inc, Sun Microsystems, Microsoft. August 2002
19. Apache Axis, Apache Software Foundation. http://ws.apache.org
20. Jakarta Apache Tomcat, Apache Software Foundation. http://jakarta.apache.org/tomcat
21. JDOM, http://www.jdom.org
22. Castor, http://www.exolab.org
23. XML Spy, Altova, http://www.xmlspy.com

# Prototype of Platform Independent Editor Using Unified Modeling Language

Challapalli Venkata Vijay Chaitanya[1] and Koduganti Venkata Rao[2]

[1]III/IV Computer Science and Information Technology
[1] chaitanya_vijay@rediffmail.com
[2] vrkoduganti@co.uk
[1,2] Gayatri Vidya Parishad College Of Engineering,
Accredited by National Board of Accreditation,
Affiliated to Jawaharlal Nehru Technological University,
Madhurawada, Visakhapatnam,
Andhra Pradesh, India. Pin – 530041

Abstract: This paper describes a prototype of Editor. This prototype deals with web-based technology and occupies diminutive amount of space, which enables a team of users to work simultaneously on different systems with the same Editor. This has the capability of running in the server and client system independently. More over this Editor have achieved the goal of platform independency and it has it's own capabilities to run on different platforms. Along with these features this also have the feature to send messages to the server from the client.

To enable concurrent access and platform independency the editor uses some of the concepts of JVM, html, UML, and scripting languages. These features are used to facilitate editing and operating. With the help of JVM the messages are being sent to the server from the client.

## 1 Introduction

Operating systems play a major role in computing. Every operating system has its own advantages and disadvantages. But it is not fair to restrict a computer intellectual to a particular operating system, since the trend in the technology always varies from time to time. With the improving technology one cannot survive by having knowledge of only one operating system. But as the operating system changes it is becoming more and more difficult for a particular individual to have an idea of all the applications present in the operating system. Hence if the applications are made in such a way that they are platform independent then one can easily handle those applications and execute them to have their corresponding results. Here it decreases the necessity to remembering the name of the same application in different operating systems. Hence it is now required to have an application or software that can be run on any platform. More over it would be more useful if the same application has the features of forwarding the messages to the server from the client so that the server-client relationship can also be achieved through the same application.

The prototype described in this paper packs all the above features and helps the user to operate on any platform with the same application. This application mainly

deals with the text and the graphics modes (formats) of operation and has the capability of handling the text files and the graphic files.

Text files are handled in this editor using the concepts of html and scripting languages and these can be stored in the client system.

Graphics files are typically handles in the system. These files are handled using the concepts of JVM and html.

UML plays a major role in the design issue and every part of the system is designed in the UML and the diagrams in the UML play a key role in the design issue of the entire prototype.

# 2 Example Scenarios

## 2.1 Example 1

The editor in the DOS operating system is restricted only to that of the DOS and this can be opened by the command “EDIT”. As this opens, one can type the corresponding file and can store it. This editor has it’s own advantages and disadvantages. The main disadvantage of this editor is that it cannot handle the image files. This command individually occupies a total space of 80692 bytes.

## 2.2 Example 1:

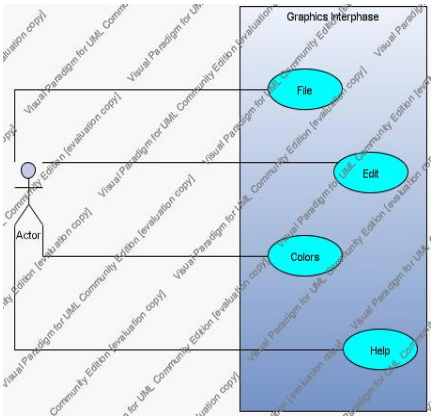
The editor in the WINDOWS operating system is “Notepad”. This also acts in the similar fashion to that of the general dos editor have almost the same options. For image files one has to open the “Paint” in windows and continue with his work. One cannot have the concurrent access of Notepad and Paint from the same application and they as a whole occupies a disk space of 73,728 bytes.

# 3 System Overview

As mentioned above the Editor described in this paper is completely web-based and browser running Editor. This Editor is a composite of both text mode of input and graphic mode of input. The Editor is likely to start at a page where the user is given a choice to choose whether he wants to go for the text mode of input or the graphic mode of input.

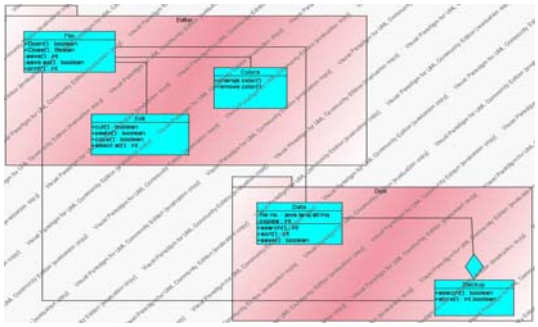
With in the project each user is given permissions to

- Create a new file
- Edit the files
- Change the colors of the background
- Add figures to the files.



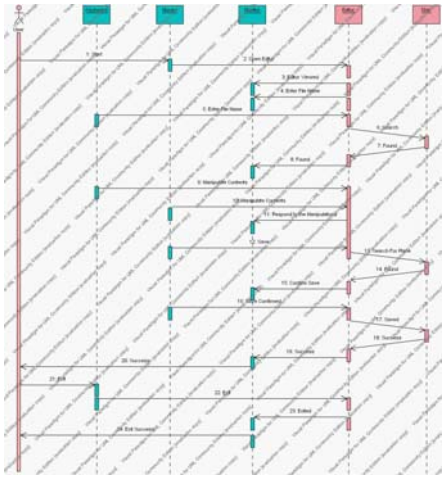

The original user deals with the file use case. Here he opens a new file and has permissions to input the text or to draw the figures. The edit use case helps him to edit the contents of the file and the colors use case helps him to add colors to the total project and the help use case to provide necessary help to the user. The user can directly interact with the graphics editor package, which provides all the above-mentioned use cases and can draw the required figures or to input the required text for him.

The general architecture or the class diagram of the above mentioned graphics editor have the corresponding operations to handle the files and their objects. Here the class diagram consists of the methods such as open, close, save, save as, exit for the classes such as file, edit, etc., This diagram mainly deals with the major operations of the file handling and achieves the total concept of the project.

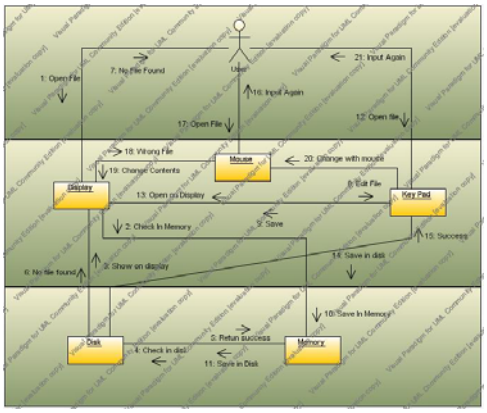


The major design issue lies in the sequence, which the total editor follows. Here the total sequence is to be taken care of and have the capability of representing the total diagram in the single sequence diagram. Our editor directly opens through any platform and has the capability to ask the user about the operation he wants to use and the mode in which he wants to enter the editor. Here he is given a chance to select the options. Since the complete user interface is designed in html the user even without having a minimum knowledge of the computer can also easily access the complete system. Then he can select the respective option and can follow the mentioned respective operations on the screen and can easily handle the complete software. The

sequence diagram is as shown in the figure and user can directly handle the respective operations and can achieve the result.



The design issue also lies in the collaboration diagram where one can view the total concept in the top-down approach and he can realize the total concept laid inside it and the path through which the messages take the choice, which the user opts. Here the objects of the same category lie in the same swim lane and this helps the user to identify easily where the object belong to the which category and he can come to a final understanding where he can directly interact with the system more easily and conveniently and effectively. The more the user gets into the details of the system the more he can adjust with it.




## 4 Discussion

Although the development of the prototype presented here is still going on it would be useful to review the benefits and the limitations of the Editor.

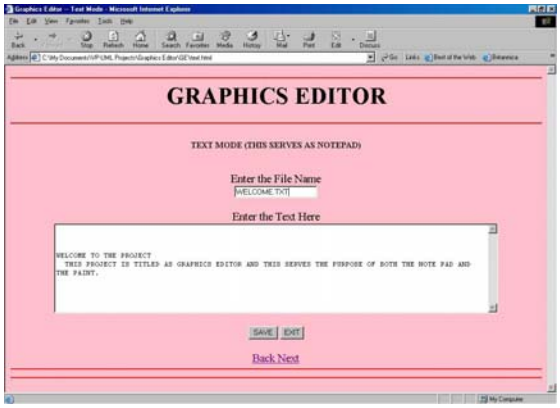
### 4.1 Benefits

This Editor has its unique benefits, which are not present in the other editors.

This is completely a web-based editor and hence all the benefits that are provided in the browser can be directly used in the editor with out any limitations.

This provides a chance to use the editor and the Internet at one instance of time since this is completely a browser-based editor.

This has the capability of forwarding the messages (graphical) to the server directly (only when the net is connected) through this editor and hence message passing becomes so easy.




Since the Editor is completely designed using the html and JVM, bagged up all the features of both of these, this have the capability of running in any operating system with out the intervention of the operating system.

The major benefit in this is that the operating system (i.e., on which it is running) only provides maximum options to it and hence the designer of the system does not have to spend more time in providing all the options to it.

This provides the most user friendly nature and this can be operated by any user who do not have that much knowledge about the computer as well as the operating system especially.

### 4.2 Limitations

Because of the editor is designed completely in the browser there are certain limitations to it.

We have first designed a prototype so that the original project time decreases and the prototype can increase the understandability of the user.

Since our Editor is a web based editor we cannot have the page numbers and other additional features with cannot be available in the browser.

Present system saves the text files only in the .txt extension.

It is limited only to create graphics files and is not capable of modifying the already created files, which are present in the system.

Since we have used the concept of applets in JVM, it is not capable of saving the file in the local machine.

This system does not support any paragraph counting, word counting and spell checking functions.

This is the complete overview of the system and this system has it own benefits and the limitations and more over there is so much to enhance in this system. The future enhancements are discusses below.

## 5 Conclusions and Future Works

Having implemented this prototype of a Editor which deals with the text and the graphics formats we are confident that this achieves some of the major goals of the present trend such as platform independency, less space utilization on the disk, more reliability, more user friendly and combined features of several applications. We can also have a look to the future works.

It is planned to implement the complete features of the WORD such as formatting, macros, tables, etc., in the Editor, which helps the user to edit his text completely.

An import function helps in importing the graphics file and helps in manipulating the contents of the graphics file. This can help the user more and the usage of the system improves.

Last but not the least: It is important to integrate these systems into the daily teaching and research practices to gain more experience and more impulses for the further developments. It should become a general practice to use these systems in the daily life to improve more and more.

## References

1. The Unified Language User Guide by Grady Booch, James Rumbaugh and Ivar Jacobson
2. Real Time UML by Bruce Powel Duglass
3. Analysis Patterns by Martin Fowler
4. Surviving Object-Oriented Projects
5. Object Solutions by Grady Booch
6. UML Distilled by Martin Fowler
7. Software Reuse by Ivar Jacobson
8. The Unified Software Development Process by Ivar Jacobson
9. Visual Modeling with Rational Rose and UML by Addison-Wesley Object Technology Series
10. Software Project Management by Walker Royce
11. The Unified Model Reference Manual by James Rumbaugh
12. Applying Use Cases by Geri Schneider
13. The Object Constraint Language by Jos Warmer
14. Enterprise Computing with Objects by Yen-Ping Shan
15. http://user-mode-linux.sourceforge.net
16. http://www,onesmartclick.com
17. http://www.tigris.org
18. http://hotscripts.com

# Service Oriented Model Driven Architecture for Dynamic Workflow Changes

Leo Pudhota, Elizabeth Chang

School of Information Systems, Curtin University of Technology,
PO Box U1987, Perth WA 6845, Australia
{PudhotaL,ChangE}@cbs.curtin.edu.au

**Abstract:** Collaborative workflow management systems in logistic companies require strong information systems and computer support. These IT integration requirements have expanded considerably with the advent of e-business; utilizing web services for B2B (Business to Business) and P2P (Partner to Partner) e-commerce. This paper proposes service oriented model driven architecture for dynamic workflow changes and strategy for implementation of these changes by isolation of services and business processes where by existing workflow systems can easily incorporate and integrate the changes following a step by step process replacement synchronization in workflow. This paper describes conceptual framework for prototype implementation resulting in dynamic collaborative workflow management.

## 1 Introduction

In this paper we discuss the design of workflow management system for dynamic business processes of large logistic consortia. Often we see that the business processes are composed of several parts, a structured operational part and an unstructured operational part, or they could be composed of semi-structured parts with some given and some unknown details. Unpredictable situations may occur as a result of changes in decisions made by the management. The inability to deal with various changes greatly limits the applicability of workflow systems in real industrial and commercial operations. This situation raises problems in workflow design and workflow systems development. We propose workflow implementation through service oriented architecture and system isolation for making changes to the existing workflow.

## 2 Dynamism In Collaborative Workflow

The advent of the web is to bind organizations together, for carrying out sales over great distances and at any time has created new modes for marketing and enabled partnerships, previously inconceivable within a wide array of businesses, as well as other human activities [1]. This IT support has expanded with the advent of e-commerce. However, with this advancement of B2B (Business to Business) and P2P (Partner to Partner) e-commerce [6], there has been an increasing tendency to set up

consortia that represent several players in a given field. Such consortia consist of companies or organizations in a given field that get together and produce a single site or what appears to be single site in order to increase traffic through the site compared to other competitor's sites and/or extend beyond their region of operation, Collaborative workflow management systems of a business sector like logistics consortium with multi-users and very dynamic environments will have: workflow *specification*, workflow *execution*, workflow *evolution*, workflow *auditing*, transaction management, workflow recovery, workflow interaction (for cooperative work), and others. The specification of a workflow consists of three items: *Process, Data, Invocation.* Changes in a workflow may be an every-day routine in a working environment. Such changes are of three types: *Modification*: new workflow has same objective but different logic and replaces old one. *Versioning*: as before but new workflow does not replace old one, but co-exists with it. *Extension*: new workflow has different objective and therefore additional logic and replaces old one.

In addition, some environments require dynamic rather than static workflow evolution, i.e., changing one part of the workflow while another part is running.

## 3 Web Services For Collaborative Logistic Workflow

A Consortium consists of many departments; generally there are six operational divisions: Management Department, Warehouse Department, Logistic Department, Accounts Department, Customer Service Department and Transport Department. Each department has its own responsibility. However they are connected to each other. Warehouse Department now already has its own system, so does Accounts Department. The complexity of works become bigger and bigger when the customer's order increase. It is hard to know the progress of the orders and warehouse check. It is also difficult to schedule the trucks, manpower, etc. Consortium likes to change its internal work (flow of works among department) and its external work (flow of works with its customers and other collaborative organizations). Consortium would like to integrate various departments, and also with other logistic network companies in its consortium. Consortium also wants its customers to be able to book warehouse service, logistic service, place orders and view the status of orders, etc on the internet. This is more like e-commerce way [3]. Here sellers are logistics providers, buyers are customers and web services brokers are web services integrators, Consortium is interlinked through internet and services are provided by web services. The basic premise behind Web Services is that a piece of code is made available to remote machines, using specific protocols, over the Internet. The Service part of Web Services relates to the idea of providing access to functionality without having to download or install the code, and the Web part refers to the means through which the functionality is accessed [19]. The three component standards of Web Services are the Simple Object Access Protocol (SOAP), Universal Description, Discovery and Integration (UDDI), and Web Services Description Language (WSDL) [19].

## 4 Issues Of Dynamism In Collaborative Workflow

Activities and artefacts do not quite constitute a process. We need a way to describe meaningful sequences of activities that produce some valuable result, and to show interactions between processes. Changes in collaborative workflow have to be incorporated into the integrated enterprise system. [8, 9, 10, 11].

In this paper we are concentrating on, 1. Design and Implementation of integrating solution for adaptation of changes in the new workflow into an already existing workflow. 2. Synchronization of new workflow to existing workflow. Other issues like Management of data scattered over multiple origin systems/legacy systems, for example, a company will have consolidate data in one logical view with a unified architecture, thereby enabling data-source independence. Because application data continues to live and change in the origin systems, the new software layer must be able to retrieve origin data on the fly and also propagate changes back to the origin systems [17]. Provide support for transactions/interaction across multiple back-end systems. These issues will help in having a uniform data processing environment for the whole enterprise, which would lead to changes and improvements in customer services, control of receivables and increase efficiency in communication, sales, marketing as well as minimization of warehouse stocks, streamlining inventory and logistics flows. Provide control to Consortium management to monitor the collaborative enterprise's condition, its stock, order and its general financial condition on a routine basis, This is indispensable to the management processes and enhances decision-making and changes which need to be taken on the short term and long term bases for the consortium to compete in the global market.

## 5 Service Oriented Framework To Support Backend Collaborative Workflow

In this paper we present a service oriented framework for collaborative logistic companies. The framework is divided in 3 sections 1. Business web services layer. 2. Services communication layer and 3. Process and transaction layer. In Business web services layer browsers interact with HTTP servers in their normal way taking advantage of any technologies that enhance this browser-to-web server link [7]. Enterprise model framework shown in figure 1, balances across one or more application server processes (also called instances) running on one or more machines. Once running, Enterprise service framework instances do not go away between user requests; they maintain themselves, their session's state for users, and their database connections. They are efficient, fast, and by definition redundant. It's the job of the HTTP server adaptor to communicate with a given HTTP server and forward requests to one or more application "instances" - an instance is a separate copy of a given application process. Enterprise services framework serving a few users may have only one instance.

A large application may have tens or hundreds of instances running on one or more machines. If an application has more than one instance, the Services controller is essentially acting as load balancing agent. If an instance fails, it only affects that

particular instance – all other instances and/or the site's web server is unaffected. The controller will forward requests over the network as easily as it will forward requests to applications running on the same box as the HTTP server. In fact, from a load sharing perspective, it is ideal for the HTTP server and Application servers to reside on separate boxes, since applications are server based, database access happens behind the firewall. Browsers need never make direct connections to a database server. Services access controls database connections so that they are highly secure (only accessible via actual application API), and conserved (that is, you never have

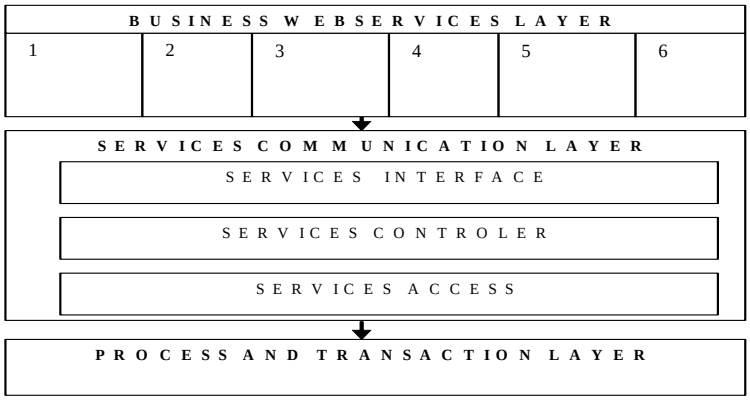



**Fig. 1.** Enterprise model Frame work

more than one connection per service unless this is specifically something the developers insist regardless of the number of users it supports). Process and Transaction layer works on underlying java foundation containing fundamental data structure, implementations and utilities used. we see each department has its own responsibility; however they are connected to each other. In a collaborative context, communication may have to be coordinated not only with in the organizations but also across organizations. Therefore a consortium may require synchronized coordination of activities of inter and intra organizational departments. This Conceptual Model provides an architectural separation of business functionality from technology implementation. This separation allows designers to use business rules defined in a UML model to drive two distinct steps in implementing such systems.

Step1. Create platform independent models in UML. The first model is a generic domain model, used to build a common understanding and vocabulary among warehouse Logistics domain experts. Step2. The domain model is then mapped into a representing warehouse logistic business. Each of the models includes a detailed set of UML Class Diagrams, Use Cases and associated Activity Diagrams describing the system [12].

# 6 Conceptual Model Of Services Communication, Process And Transaction Layer

This logical architecture of the web services frameworks is a programming building blocks of the largest granularity. Web services Frameworks is responsible in providing application's user interface and state management. Since applications are server based, database access happens behind the firewall. Browsers need never make direct connections to a database server. Services access controls database connections so that they are highly secure (only accessible via actual application API), and conserved (that is, you never have more than one connection per instance regardless of the number of users supported - unless this is specifically something the developers desire). Designers can use business rules as defined in previous section to define in UML model. Using this business model, we can create one or more subsystems to represent the logical functions of each of the enterprise systems. This business model contains both the details of the business logic, as well as the mapping of the logic into the major subsystems. The business model forms the basis for managing all changes to the current systems. And the next step is System Integration using Conceptual Model of Platform Specific Models (PSM's), for each of individual systems to form enterprise system [12]. These models were each derived from one or more subsystems in the business model. System construction consists of customizing each of the enterprise systems, and creating the business logic. Business logic that spanned systems is constructed using components technology and deployed in the application server also called web service brokers.

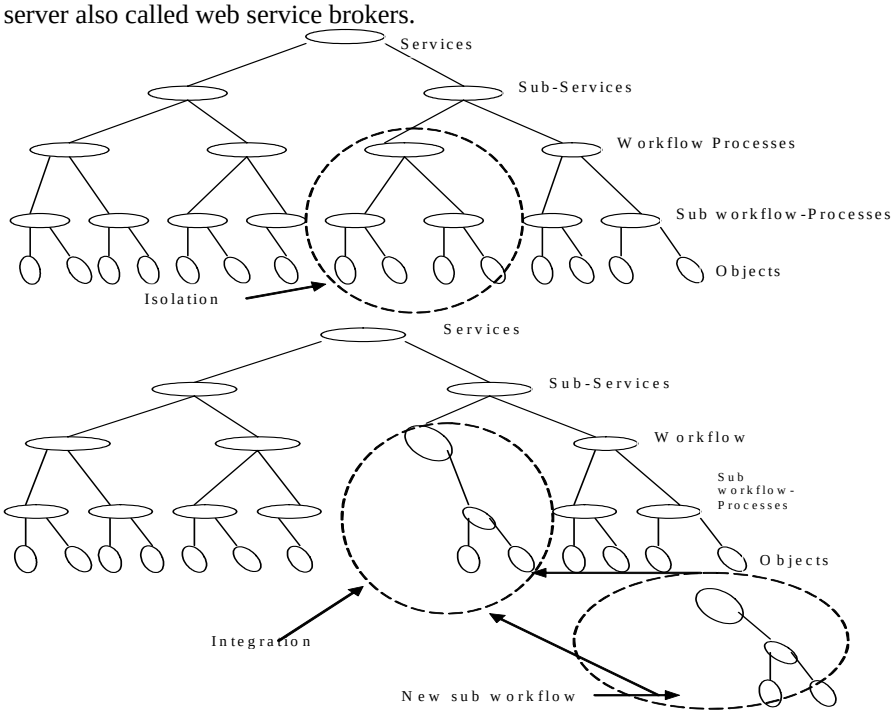



**Fig. 2.** Isolation replacement and integration of new workflow

Process and transaction framework contains an underlying java Foundation made of fundamental data structure implementations and utilities used throughout the rest of Enterprise processes. Examples include arrays, dictionaries and formatting classes. These processes provide RDBMS independence for services persistence, provides object persistence transaction management, and provides services useful for web based presentation and deployment. It also provides an environment to use and create reusable components; it facilitates the use of true business objects in services oriented framework and handles storing and restoring objects to a data store and usually in a relational database. Since the business processes and objects created don't care about the underlying database or how their values are presented in user interfaces, they may be re-used over and over in any number of different web applications and can be maintained by developers. Web services framework also provides a persistence layer to maintain information at all time.As a single process can produce a huge workflow map, the *subworkflow layers* allow the workflow to be broken down into more manageable sections. This also allows modularisation of commonly used functions – for example bulk notification activities rather than having them repeated throughout the main map or even several workflows maps or systems. This also makes them easier to manage and maintain. Sub workflow layer can be very useful to split your main process into its constituent elements – in a large process there is likely to produce a quicker initiation and processing. However, in some cases, the overhead of moving from a 'parent' workflow into a 'child' sub workflow can be a lot higher than the performance benefit of doing so, hence we need to plan the workflow carefully. This type of architecture will help in bringing about main areas of changes like: Services Layer changes like criteria determining field colouration or visibility or edibility of a given field has changed, or a popup box is now required if some criteria is met.

- Data Changes / Mass Updates like Value Added Tax calculations need to be redone if VAT rate changes, customer name changes
- Business Logic changes like Logistic criteria changes, routing requirements change.
- Patches / Bug Fixes that need to be applied to many active workflows

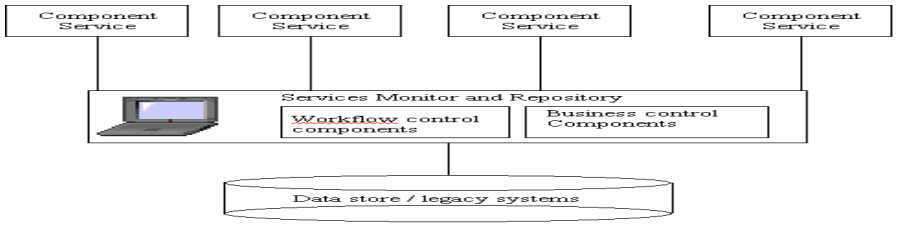



**Fig 3.** Implementation Framework

One approach is to suspend, correct and restart each workflow in sequence, although for large numbers of workflows this would be very time consuming [18].

# 7 Frameworks For Services Components And Implementation

We propose use of a modular approach to software development for implementation of this model, current advancement in technology has resulted in better quality, reusability, productivity, and cost effectiveness. Changes to the system composition and configuration are limited once the system has been compiled. In order to solve the complexity and flexibility issues of large-scale software, We encapsulate business logic in a workflow, and use component based middle layer called Services Monitor and Repository which acts as a centralized server that contains all diagrams, reports, forms, data structure, data definitions, process flows, logic, and definitions of organizational and system components; it provides a set of mechanisms and structures to achieve seamless data-to-tool and data-to-data integration, this middle ware provides the link between Component Services and data store. This services monitor and repository layer follows strict object oriented principles, it contains two major parts, workflow control components and Business control components which contains all the objects that execute the complex business rules. In data store large complex workflow processes are broken down into smaller workflows and sub *workflow layers* to be able to better manage and maintain each section. Some process activities may be repeated throughout the main map or even several workflows maps or systems. This allows modularization of commonly used functions and help in easy management by Services Monitor described in detail below. Data store shown in Figure 2 aims to eliminate latency by allowing multiple applications to access a single physical data store directly. This architecture is suitable when applications and databases are located in the same data centre; this approach is more intrusive because we usually have to modify some applications to use a common schema. Reading data directly from a database is generally harmless, but writing data directly into an application's database risks corrupting the application's internal state.

Although transactional integrity mechanisms protect the database from corruption through multiple concurrent updates, they cannot protect the database from the insertion of bad data. In most cases, only a subset of data-related constraints is implemented in the database itself. [20]. To avoid this we include Services Monitor which is visual tool mapping software which is part of the services monitor and repository layer, we use component technology for data management in order to extract the underlying schema in the datastore which is also in the form of components. Services monitor allows us to identify the workflow processes and sub workflow processes and objects stored in the data source, which need to be isolated and a new sub workflow which has to be integrated, it also helps to create, edit, or delete existing data store objects dynamically when connected to the datastore. We can interact with the server data store using datastore diagrams incorporated in the service monitor. Datastore diagrams graphically represent the tables as of a normal database. These tables display the columns they contain, the relationships between the tables, and indexes and constraints attached to the tables. We can use data store diagrams to: View the tables in your database and their relationships. Perform complex operations to alter the physical structure of the database.

We can make changes freely in the datastore diagram without affecting the underlying datastore. When we modify a datastore object through a datastore diagram, the modifications made are not saved in the datastore until we save the table or the

datastore diagram, Visualize the structure of your database tables and their relationships. Provide different visualizations of complex databases. Experiment with database changes without modifying the underlying database. Create new tables, indexes, relationships, and other constraints. Alter the structure of your database. Thus, we can experiment with "what if" and various workflow scenarios and also check if these changes made to the workflow can be integrated to the existing workflow, using a datastore design without having to permanently affect its existing design or data. During editing, we can experiment with different object definitions to see if proposed modification will affect the datastore. When we complete these modifications, we can either save our diagram/design or update the database to match the diagram, or we can discard it leaving the underlying database unchanged.For example [please refer to figure 4], you can create/view a database diagram for customer services department that shows only tables that hold local delivery of goods information. We can view workflow part of the process that shows only those tables that are used in this specific workflow module, here we can make change to Devanning process to replace Warehouse code with Warehouse type code and Delivery docket with Delivery time. We can change the size, shape, and position of objects in the diagram without affecting their definitions in the database. When we save the datastore diagram, the layout of the diagram is preserved as well as any changes made to the object definitions in the diagram are also saved.

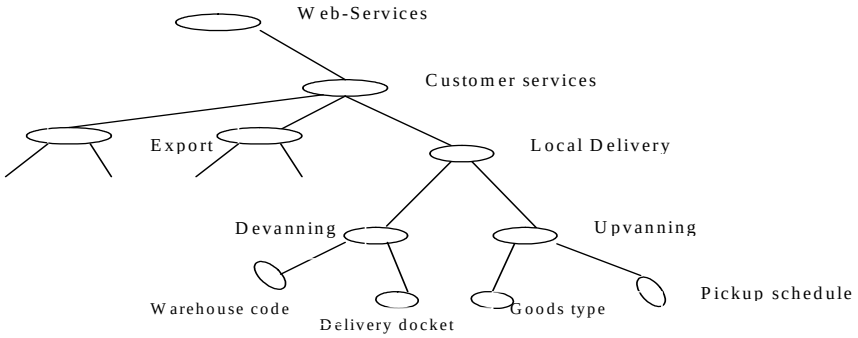



**Fig. 4.** Example of local delivery process

So as to keep the whole consortium process running we propose exclusive locking mechanism; the locking level Performance and concurrency can also be affected by the locking level used, Exclusive locks are exclusive to the user till the changes are made without having to dissturb the overall workflow. Exclusive lock on a record means that part of the process is denyed access, there by that part of the workflow is isolated so that the required changes can be made only to that part of the process, one may choose some objects or even all of the workflow or sub workflow tasks to be associated with implicit invocation. determines the size of the process that is locked. [21]. This framework is applicable to a diverse range of software Performance and concurrency can also be affected by the locking level used, Exclusive locks are exclusive to the user till the changes are made without having to dissturb the overall workflow. Exclusive lock on a record means that part of the process is denyed access, there by that part of the workflow is isolated so that the required changes can be made

only to that part of the process, one may choose some objects or even all of the workflow or sub workflow tasks to be associated with implicit invocation. When editing of table in a datastore diagram has been done, an asterisk (*) Example Delivery time *, appears after the table name in the title bar to indicate that the table contains changes to the workflow that have not yet been saved in the database. This indicator appears as a result of a change made to the workflow objects in the datastore, represented as a column or index, in the table of the diagram/ design. When we add a modified table to another open diagram, the table appears there with its unsaved changes and an asterisk in its title bar. When you save the table or the diagram, the asterisk disappears.

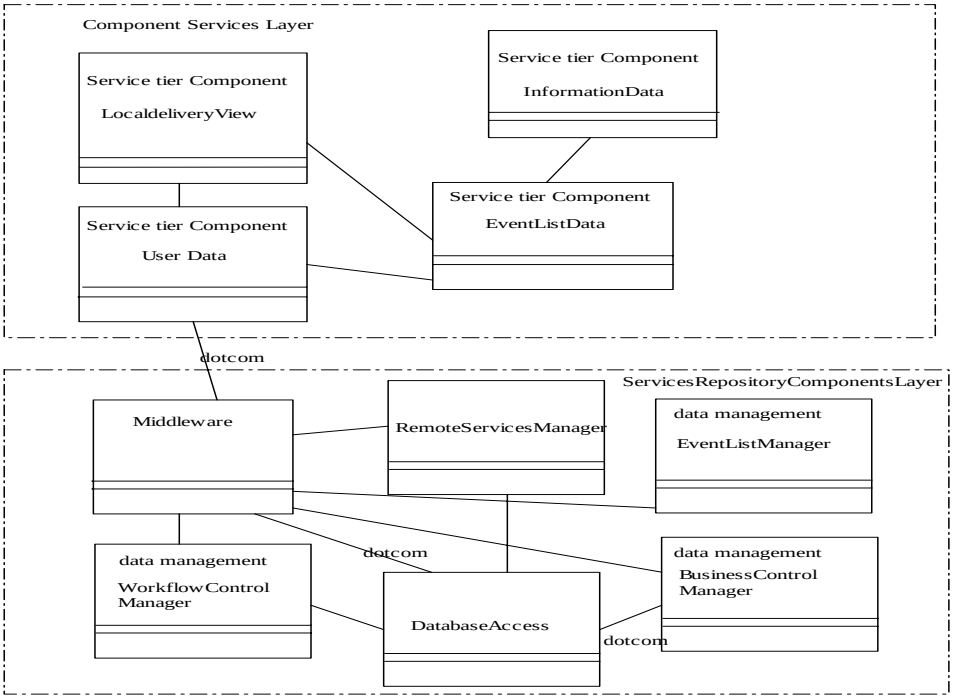



**Fig. 5.** Example of component design for our services

This reconfigurable plug and play object component-based framework [please refer to figure 5] is used to specify collaborative software construction, customization, integration and evolution through the reuse of context-independent objects where composite architectures are hierarchically constructed from layered groups of collaborating component plug-ins development environments.

For user interface web services layer browsers interact with HTTP servers in their normal way taking advantage of any technologies that enhance this browser-to-web server link. For example secure socket layer communication protocols in Netscape and Microsoft browser/server products browsers communicate with HTTP servers, which communicate with the Application Server. The Business web services layer generates web application at run time, Services communication layer provides application's user interface, state management and provides an environment to use

and create reusable components [7]. business logic is separated using two tier approach, web services are generated at runtime from metadata in the services repository component layer, this middleware data describes the webservices and its interaction with the underlying business logic components [22]. This procedure helps us to have multiple user services configuration based on shared business and workflow logic components.

## 8 Conclusion

In this paper, we have discussed service oriented architecture for dynamic workflow systems. We have also discussed issues and frameworks, service oriented enterprises systems and have come up with approach for dynamic adaptation of the changes to the existing workflow. We propose implementation of such systems by the process of isolation, integration and synchronization, our future research will be to implement this plug and play software development methodology and come up with a working prototype of our system.

## References

1. Marshak, R.T 1994.: “Falling in Love with Distinctions”, In “New Tools for New Times: The Workflow Paradigm”, Future Strategies Inc.,
2. Miers, D 1996: “The Workware Evaluation Framework”, Enix Limited,
3. Chang, E 2000: Requirement Specification of Logistic Manager for Software Engineering Project, Department of Computer Science and Software Engineering, The University of Newcastle,.
4. Haake, J.M., Wang, W. 2002: “Flexible Support for Business Processes: Extending Cooperative Hypermedia with Process Support”.
5. Denning, P.J.,1994 “The fifteen level”, In Proceedings of ACM SIGMETRIC Conference on Measurement & Modeling of Computer Systems.
6. Sheth A 1996..: “State-of-the-art and future directions”, In Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems.
7. David Neumann, 2004. An Introduction to WebObjects. Retrieved:July30,from http://.mactech.com/articles/mactech/Vol.13/13.05/WebObjectsOverview.
8. Pudhota L, Chang E. et al 2003,. International Journal, Computer Science, System and Engineering, **“**Extension of Activity Diagrams for Flexible Business Workflow Modeling “volume 18 no3 UK.
9. Pudhota L, Chang E 2004 “collaborative workflow management for logistics consortium” ICEIS Porto, Portugal.
10. Pudhota L, Chang E 2004 “Modelling the Dynamic Relationships between Workflow Components” ICEISl Porto, Portugal.
11. Pudhota L, Chang E, Venable J 2004 “E- Business technology adaptation through workflow mining” MSV Las Vegas, Nevada, USA.
12. http://www.omg.org/mda/mda_files/UNextMDA4.pdf
13. Ulieru. M, Robert W. Brennan Scott S. 2000 “The holonic enterprise: a model for Internet-enabled global manufacturing supply chain and workflow management”, Canada.
14. Ulieru. M, Stefanoiu. D, et al2000.. “Holonic metamorphic architecture for manufacturing” University of Calgary, Calgary, Canada

15. Brandenburger, A. M. and Nalebuff, B. J., 1996, Co-operation, Doubleday NY.Brennan, R. 2000, “Performance comparison and analysis of reactive and planning-control architectures for manufacturing”, Robotics and Computer Manufacturing 16(2-3), pp. 191-200.
16. Christensen, J.H, 1994 “Holonic Manufacturing Systems: Initial Architecture Standards Directions”, Proceedings of the First European conferenceManufacturing systems, European HMS Consortium, Hanover, Germany.
17. http://www.journee.com/n_hl_020703b.html
18. Anastassia.A, Yannis E. et al 1998 “Scientific Workflow Management by Database Management” book title = "Statistical and Scientific Database Management" pages = "190-199",
19. Clark, M., Fletcher, P., et al 2002. Web Services Business Strategies and Architectures. Expert Press.
20. Talevski, A., Chang, E., Dillon, T., 2003 “Overview of a Plug and Play Component-Based Framework”, World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, USA.
21. Maloney, J., “COM Reality Tour”, Microsoft Cooperation. On-line at: http://www.microsoft.com/com/ August 1997
22. T. Andrews, F. Curbera etc.: Business Process Execution Language for Web Services Version 1.1. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, May 2003.

# Design and Prototyping of Web Service Security on J2ME based Mobile Phones

Ti-Shiang Wang

Nokia Research Center
5 Wayside Road, Burlington, MA 01803, USA.
ti-shiang.wang@nokia.com

**Abstract**: One of the main objectives in this paper is to investigate how to manipulate the Simple Object Access Protocol (SOAP) message and place security functions in the header of SOAP message. Here, we will present the design and implementation of web service security application on Java 2 Micro Edition (J2ME) based mobile devices. Basically this prototyping includes two-stage approach. In the first stage, we study the concept of proof in implementation of web services security on the IBM laptop using IBM WebSephere Studio Device Developer (WSDD V 5.6) IDE [1]. In addition we import kXML/kSOAP APIs to process SOAP message and use Bouncy Castle's API [2] supporting cryptographic algorithms for security implementations. In this paper, the security functions we present here include five tasks: non-security, data digest, data encryption using symmetric key, data encryption using asymmetric key, and digital signature. At each task, we will discuss its corresponding design, SOAP header message, time performance, and return results in emulator. Based on the expected results from the first stage, in the second stage, we use Nokia 6600/3650 mobile phones as target mobile devices to test our application and evaluate performance at each task. Finally we will share our experience and lessons on this work in the conclusion and do the demonstration using Nokia 3650 mobile phone in the conference.

## 1 Introduction

As mobile phone becomes a commodity that almost everyone will own one mobile phone but may not have the traditional landline, it is very likely that the mobile phones will replace PDA (Personal Digital Assistant) devices in some applications. The web services are services provided over Internet or intranet using standardized XML messages to exchange information among different nodes. In addition, web service is not tied to any platforms or programming languages, which may need extensive technical skill. Also, eXtensible Markup Language (XML) remote procedure calls (XML-RPC), SOAP [3] or even HTTP can be used to implement the messaging of web services. On top of the transport methods, web services use Web Services Description Language (WSDL) to define the service provided by service applications so requester and application provider can communicate each other regardless of programming language or platform. Because web services are self-describing, discover-

able, and independent to any platforms, it can support automated application integration and help to improve the development process.

To illustrate the thoughts of implementing web service security, IBM WebSephere Studio Device Developer (WSDD V 5.6) IDE, which is a J2ME development tool, which supports automated stub generator and other advanced features, is considered in this paper. That is, we use IBM WSDD to generate prototype files called "*stubs*" and continue developing codes based on the generated files. The "*stubs*" are generated based on WSDL file from remote server. It contains the methods to process necessary parameters and arguments to access remote services. The "*stubs*" may not have complete codes but it serves as a base for further development. The ultimate goal is using web services to build an application-centric web, which has less human interaction involved. Thus, in this paper we will only focus the discussion on client-server web services security implementation rather than enterprise web services, which will be part of our future works. For manipulating SOAP message, though JSR 172 web services specification also supports access to remote SOAP/XML based web services and parsing XML data on the J2ME platform [4], it is not possible for J2ME mobile devices with limited processing power to include all JAXP functionalities. In addition, current JSR 172 specification does not support SOAP message header handler.

The kXML [5][6] is a project to provide XML pull parser for J2ME based mobile devices. It supports XML namespace, and XML writing. These APIs have ability to process SOAP message using XML parser engine from kXML. kXML/kSOAP API (an open-source J2ME XML and SOAP parser). In this work, both kXML and kSOAP have to be included in the java classpath to provide the functionalities of process SOAP messages. To implement the security functions in the SOAP header, W3C has suggested adding these security tags into SOAP header as its security extensions. This work will follow the recommendation of W3C to add security information into SOAP header. As far as cryptographic algorithms used for mobile devices [7] are concerned, we test and use Bouncy Castle's cryptographic API, which is an open-source Java cryptographic algorithm API for J2ME mobile devices. In this paper, five tasks of security function are to implement: no-security, digest, encryption with symmetric key, encryption with asymmetric key, and digital signature. At each task, we will show its SOAP message, demonstration of result, and time performance. For the web server we used for this demonstration, we adopt temperature web service provided by Xmethods [8][9]. As we mention, in this work, we focus on the implementations of security function at client side, that is, the mobile phone or user side. Then in the last section, we will draw a conclusion and discuss some future works.

## 2 Architecture and Implementation

In the client side, developer can use IDE or automated tool to generate stub, a prototype or template file to access web services based on the WSDL file. In previous version of IBM WSDD (Version 5.5), which supports both Document-style and RPC-style web services and the IDE can help us to generate ***Temperature_Stub.java*** file as a way to automate the application development. However, for the WSDD 5.6 version, only document-style web services are to support. Thus we use document-style tem-

perature WSDL for our implementation in the client side. Figure 1 shows the generic architecture for our work. In the first stage, we use laptop and IBM WSDD tool kit and emulator to demonstrate the concept of this implementation. Then in the second stage, we use Nokia 6600/3650 mobile phone as the client component.


**Fig. 1.** Proposed web service architecture using mobile phone

There are two modules, *cryptographic algorithm module (CAM) and SOAP message parser module (SMPM)*, required to implement web services security. The CAM includes following files:

- ***Encryptor.java***: an abstract class to define the interfaces of encryption and decryption algorithm. The "*Encryptor*" class acts as a parent class for all security classes. As an abstract class, the real implementation needs to be done after inheriting from it, so that further security extensions can be added or integrated under the "*Encryptor"* class.
- ***DigestEncryptor.java***: the implementation of data digest algorithm. This class implements the abstract method of ***Encryptor.java*** file;
- ***SymmetricEncryptor.java***: the implementation of secret key data encryption algorithm. This class implements the abstract method of ***Encryptor.java*** file;
- ***AsymmetricEncryptor.java***: the implementation of public key data encryption algorithm. This class implements the abstract method of ***Encryptor.java*** file;
- ***DSEncryptor.java***: the implementation of digital signature algorithm. This class implements the abstract method of ***Encryptor.java*** and composite ***DigestEncryptor*** and ***AsymmetricEncryptor*** classes.

For the SMPM, it includes following files:

- ***SoapEnvelope.java***: a SOAP message parser without security extension;
- ***SecSoapEnvelope.java***: a SOAP message parser with security extension;
- ***HttpTransportTest.java***: Responsible for delivery of SOAP message.

The SOAP message handler has two classes: ***SOAPEnvelope*** and ***SecSOAPEnvelope***. ***SOAPEnvelope*** is the modified version of original class from kSOAP package and the ***SecSOAPEnvelope*** adds the header and body process capability so that

security and cipher data can be replaced in the SOAP message. In addition, to interact between user/client side and server side, there are two java application files implemented as well. ***SecTemperature_Midlet.java*** is the main class for J2ME application and ***Temperature_Stub.java*** is the interface between ***SecTemperature_Midlet.java*** and other modules mentioned above. In this work, we have implemented a temperature query web services application on J2ME based mobile phone. User of the mobile application will be asked for ***zipcode*** and the selection of desired encryption algorithm. There are five different cryptographic algorithms available for selection, including no-security (Task 1), data digest (Task2), data encryption with symmetric key (Task 3), data encryption with asymmetric key (Task 4), and digital signature (Task 5).

After user enters ***zipcode*** and chooses one of the encryption algorithms, the application will take the ***zipcode*** (for example, 01803) as input and encrypt this ***zipcode*** based on the selected encryption algorithm. Then, the application will generate a cipher value and attach this value to the body on SOAP message. In addition, the cryptography algorithm name and the web services security tags will be added to the SOAP header and body. All the name spaces and XML tags in web services security have been defined in the standard of OASIS Web Services Security [10]. It should be noticed that the original ***zipcode*** is not replaced with cipher text because the existing of plaintext and cipher value can help us to verify our implementation of cryptographic algorithms and get the result (i.e., temperature degree) from the web server.

## 3 Detail Task Implementation and Results

In this section we discuss 5 tasks of our security implementation. In addition, the corresponding SOAP message, and demonstration are presented and general time performance is introduced as well.

### 3.1 Task 1:No-Security

In this task, we study what the SOAP message looks like without adding any security function using IBM WSDD as the starting point for the rest of the following tasks. The following SOAP message shows the regular SOAP message without security extension. Figure 2 shows the snapshot and results from emulator. The time of result responded from server side is ~4 seconds.

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header />
  <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <getTemp xmlns="urn:xmethods-Temperature" id="o0" SOAP-ENC:root="1">
    <zipcode xmlns="" xsi:type="xsd:string">01803</zipcode>
   </getTemp>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

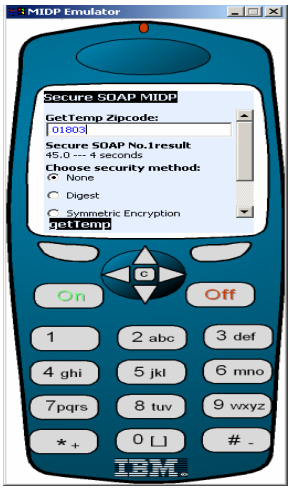



**Fig. 2.** Snapshot of no-security implementation.

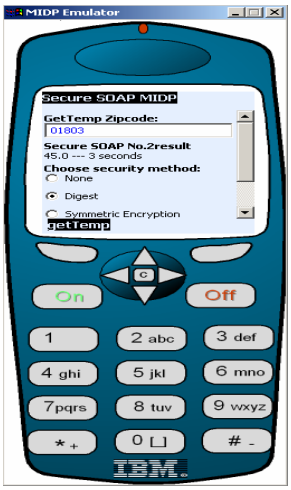



**Fig. 3.** Snapshot of data digest implementation.

### 3.2 Task 2: Data Digest

Data integrity is to ensure that the data is from original sender without any modification by unauthorized users. It is important to understand that both sender and receiver choose a key before creating or verifying the digest data. Once receiver receives the data, the digest value from the received plaintext is generated using a pre-determined key to both sides. The new digest value generated at receiver side will be compared with the digest data sent from sender side. Both of them should be the same, otherwise the data sent from sender side possibly have been modified. The following SOAP message illustrates the SOAP message with data integrity security extension.

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
  <wsse:Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd">
   <ds:DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#sha1" Algorithm="SHA256" />
   <ds:SignedInfo xmlns="">
    <ds:CanonicalizationMethod Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
    <ds:Reference URI="#zipcode">
     <ds:Transforms Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
    </ds:Reference>
   </ds:SignedInfo>
  </wsse:Security>
 </SOAP-ENV:Header>
 <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <getTemp xmlns="urn:xmethods-Temperature" id="o0" SOAP-ENC:root="1">
   <zipcode xmlns="" xsi:type="xsd:string">01803</zipcode>
  </getTemp>
```

```
    <ds:CipherValue
xmlns="http://www.nokia.com/nrc/boston/security/">d085119a2d49e7099ebf9f3fd5801bf9bebbaf77
    b2be07805577cec7598b9aa1</ds:CipherValue>
     </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
```

In this SOAP message, several lines have been added to the SOAP header. The ***<wsse:Security>*** defines the standard of the web services security for this application. The ***<ds:DigestMethod>*** defines the cryptographic algorithm used in this task. The XML tag ***<ds:Reference>*** is used to indicate what data will be encrypted from the SOAP body. Receiver is able to use this tag to re-construct the original plaintext. Here, we only encrypt the ***zipcode*** but not the whole body of SOAP message. There are also some lines adding to the BODY of SOAP message. The ***<ds:CipherValue>*** is the data digest value calculated after entering zipcode by the user. In this case, the SHA cryptographic algorithm is implemented. Please note that the digest data is placed outside ***<getTem>*** tag because there will be no response if we insert other data into the tag defined by WSDL to receive request information. Figure 3 shows the result we get from WSDD emulator and the time to receive the result is ~ 3 seconds.

### 3.3 Task 3: Data Encryption Using Symmetric Key

Symmetric encryption takes plaintext as input and use secret key to encryption the plaintext to a cipher text. In this project, we implemented the AES encryption, which has 128-bit block size of plain text. Compared with previous data digest algorithm, this task experiences more complicated since padding issue on the input message and the key needs to be considered. The following SOAP message illustrated the implementation of symmetric encryption in SOAP message.

```
    <SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
     <SOAP-ENV:Header>
      <wsse:Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
    -wssecurity-secext-1.0.xsd">
       <ds:DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#sha1" Algorithm="AES" />
       <ds:SignedInfo xmlns="">
        <ds:CanonicalizationMethod Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
         <ds:Reference URI="#zipcode">
          <ds:Transforms Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
         </ds:Reference>
        </ds:SignedInfo>
       </wsse:Security>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
       <getTemp xmlns="urn:xmethods-Temperature" id="o0" SOAP-ENC:root="1">
        <zipcode xmlns="" xsi:type="xsd:string">01803</zipcode>
       </getTemp>
       <ds:CipherValue
xmlns="http://www.nokia.com/nrc/boston/security/">dec921ebadb8dbec94a1340f532a7
    ef6</ds:CipherValue>
     </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The difference between the SOAP message for symmetric key encryption task and data digest encryption is the attribute of <ds:DigestMethod> XML tag has been replaced with “AES” to reflect the change of cryptographic algorithm. Also, the cipher value in the body of SOAP message is replaced with corresponding cipher data. Figure 4 shows the snapshot of symmetric key encryption application on emulator. In this case, ~3 seconds is required to the result sent back from the server.


**Fig. 4.** Snapshot of data encryption using symmetric key.


**Fig. 5.** Snapshot of data encryption using asymmetric key.

### 3.4 Task 4: Data Encryption Using Asymmetric Key

The asymmetric key encryption is also called “public key encryption” algorithm. Sender uses receiver’s public key to encrypt data. The encrypted (cipher text) data (here, 01803 is used) is sent to the receiver and the receiver uses its own private key to decrypt the cipher data to original plaintext. According to our test, it will take 4 or 5 minutes to generate the key pair on Nokia 6600/3650 mobile device, even though the time required in the emulator is much shorter (~10 seconds), as shown in the Figure 5. The following SOAP message illustrates the implementation of asymmetric encryption in SOAP message.

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
  <wsse:Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd">
   <ds:DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#sha1" Algorithm="RSA" />
   <ds:SignedInfo xmlns="">
   <ds:CanonicalizationMethod Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
    <ds:Reference URI="#zipcode">
     <ds:Transforms Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
    </ds:Reference>
   </ds:SignedInfo>
  </wsse:Security>
```

```
</SOAP-ENV:Header>
<SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <getTemp xmlns="urn:xmethods-Temperature" id="o0" SOAP-ENC:root="1">
  <zipcode xmlns="" xsi:type="xsd:string">01803</zipcode>
 </getTemp>
 <ds:CipherValue
xmlns="http://www.nokia.com/nrc/boston/security/">015d8dbccb65a206ccd0cee6abfe3f344a456e204e159
b11e119c48c5b0a347018263ba8341be1872cf83e58c6922a91d2758565076099583b9e84d0c946b01b425f1
d812dfc0651c40d3fc32e35bd82fd21d066d8b28eef9134dc4c60f0bcbd3c0ae0c354987aee407a3bd0cddf2e9
0d56e4f934268b93eae71406c7aa7ec81</ds:CipherValue>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

It is obvious that the more time consumption to get the result is experienced due to the limited processing power in the mobile phones.

### 3.5 Task 5: Digital Signature

In this task, both HASH and RSA algorithms are used to implement digital signature function. The original message is calculated to a unique digest value using SHA-1 hash algorithm. Then, the digest is signed by sender's private key as a signature message. Both signature message and original plaintext data are sent to receiver side. Once the receiver receives the signature message and plaintext from the sender side, the signature message is decrypted using sender's public key at receiver side. After the signature message is decrypted to a hash message, which the is used to compare with hash message generated in the receiver side using plaintext sent from sender to check the integrity of the data. The following SOAP message illustrates the implementation of digital signature.

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
  <wsse:Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-w
ssecurity-secext-1.0.xsd">
   <ds:DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#sha1" Algorithm="DS" />
   <ds:SignedInfo xmlns="">
    <ds:CanonicalizationMethod Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
    <ds:Reference URI="#zipcode">
     <ds:Transforms Algorithm="http://www.w3c.org/2001/10/xml-exc-c14n#" />
    </ds:Reference>
   </ds:SignedInfo>
  </wsse:Security>
 </SOAP-ENV:Header>
 <SOAP-ENV:Body SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <getTemp xmlns="urn:xmethods-Temperature" id="o0" SOAP-ENC:root="1">
   <zipcode xmlns="" xsi:type="xsd:string">01803</zipcode>
  </getTemp>
  <ds:CipherValue
xmlns="http://www.nokia.com/nrc/boston/security/">17638ec2a1d3a52a40ec6cd06f2242287756e84c51eb
3cb1ca75d4cd678ec92b890a92f222c8a907de81dce87caec1a1cbdf02b0d02cba5e5f9d13d30bf48f3c926222
e9d4fd568f1b1c6f01cf4933c3087427be3502f0b141d7ed70afe7364744d1af5587d7f9fb6fe11a494a3b48432
3ed403851aeccea0eae62a1edd57960
 </ds:CipherValue>
```

*</SOAP-ENV:Body>*
*</SOAP-ENV:Envelope>*

Figure 6 shows the application running digital signature function. Because it needs to generate private-public key pair as asymmetric algorithm, thus it takes more time (~17 seconds) than any one of the tasks in this paper.

.

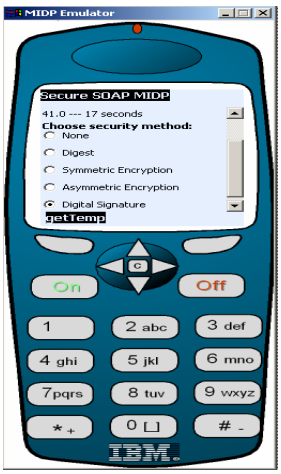



**Fig. 6.** Snapshot of digital signature.

## 4 Conclusion

As wireless networks have been widely deployed and mobile phones are popular year after year due to cost reduction, people reply on the use of mobile phone more in their daily life for specific services through the web server and Internet. It is well-known that security is one of key components that should be implemented in the web services and still remains as one of the challenging issues so far. In this paper, we have prototyped five security tasks in the client side (or mobile phone) using IBM WSDD and demonstrated these tasks using Nokia 6600/3650 series mobile phones. We also presented the corresponding SOAP message communicated between client and server sides. We also evaluated and compared the time performance of each task. Based on our design and implementation, it takes more time to generate cipher text for asymmetric key encryption and digital signature than other tasks. Due to the limited processing power of current mobile phones, when the application is running on real mobile phones, we experienced more time delay to get the result from the server side than we expected in the order of minutes. Thus how to improve the time performance at client side to meet the practical need of people is part of future work. With this time performance obtained from all the tasks in this paper, the application using web service security using mobile devices needs to consider carefully and appropriately from both technical/technology side and business side. In addition, we are also investigating the security functions implemented in the server side and planning to inte-

grate with existing results. Furthermore, some possible applications and implications using web services using mobile phones are under investigation as well.

## Acknowledgements

The author is grateful to Edmond Chang's contribution on this project when he works as an intern in Nokia Research Center. In addition, the author also thanks Dr. Senthil Sengodan and Dr. Tat Chan for their encouragement, discussion and comments on this work as well.

## References

1. IBM WSDD, http://www-306.ibm.com/software/wireless/wsdd/
2. Bouncy Castle, http://www.bouncycastle.org/index.html.
3. M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. Nielsen, SOAP Version 1.2 Part 1: Messaging Framework, http://www.w3.org/TR/2003/REC-soap12-part1-20030624/, June, 2003.
4. Jon Rllid and Mark Young, Sun Microsystems, J2ME Web Services 1.0 final Draft, http://www.jcp.org/en/jsr/detail?id=172, October 15, 2003.
5. kXML project - http://www.kxml.org
6. Enhydra.org, http://kxml.objectweb.org/project/aboutProject/index.html
7. Enterprise J2ME: Developing Mobile Java Applications, Michael Juntao Yuan, ISBN: 0131405306, Prentice Hall Publisher, 2003.
8. Xmethods.Inc, http://www.xmethods.net/, 2004.
9. WSDL files for temperature, http://www.xmethods.net/sd/2001/TemperatureService.wsdl.
10. OASIS, Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf, March 2004.

# Generating Code for Mapping UML Associations Into C#

Iraky H. Khalifa[1], Ebada A. Sarhan [1] and Magdy S. A. Mahmoud[2]

[1] Helwan University, Computer Science Department, Faculty of Computer Science and information, Egypt
{Khalifa, Sarhan} dr_iraky@hotmail.com

[2] Suez Canal University, Computer Science Department, Faculty of Computer Science and information, Ismailia, Egypt
magdy01sh@yahoo.com

**Abstract.** Object-oriented programming languages do not contain syntax or semantics to express associations directly. Therefore, UML associations have to be implemented by an adequate combination of classes, attributes and methods. This paper presents some principles for the implementation a UML binary associations in CSharp (C#), paying special attention to multiplicity and navigability. Our implementation has some specification for bidirectional associations. These principles have been used to write a series of code patterns that we use in combination with using tools which generating code for associations, such as Poseidon for UML and Enterprise Architect. These Tools are read from a UML model stored in XMI (XML Metadata Interchange) format.

## 1 Introduction

One of the key building blocks in the Unified Modeling Language [UML] is the concept of association. An "association" in UML is defined as a kind of relationship between classes (Actually classifiers. `Classifier` is a superclass of `Class` in the UML metamodel), which represents the semantic relationship between two or more classes that involves connections (links) among their instances [1]. As it has been denounced long ago [2], object-oriented programming languages express classification and generalization well, but do not contain syntax or semantics to express associations directly. Therefore, associations have to be implemented by an adequate combination of classes, attributes and methods [3,9,11]. The simplest idea is to provide an attribute to store the links of the association, and accessor and mutator methods to manipulate the links. Other approaches emphasize the use of Java interfaces to implement associations with some practical advantages [4, 13].

Poseidon UML[5] tools often provide some kind of code generation starting from design models, but limited to skeletal code involving only generalizations and classes, with attribute and method signatures, but no associations at all. The programmer has to manually write the code to manage the associations in a controlled way, so that all constraints and invariants are kept for correctness of the implementation. This is usu-

ally a repetitive task that could be automated to a certain extent. Besides, the number of things that the programmer should bear in mind when writing the code for the associations is so large, that he or she continuously risks forgetting some vital detail. This is specially true when dealing with multiple (with multiplicity higher than 1) or bidirectional (two-way navigable) associations. Moreover, the final written code is frequently scattered over the code of the participating classes, making it more difficult to maintain.

The aim of this work is three aims. **First**, write a series of code patterns that will help programmers in mapping UML associations into a target object oriented programming language. In this work, the language has been chosen to be CSharp (C#), although the principles we have followed may be applied to other close languages like C++, Java or the .NET framework. **Second** aim, using a tool that generates code for associations using these patterns, the associations being read from a model stored in XMI format. A **third** aim will be to enable reverse engineering, that is, obtaining the associations between classes by analyzing the code that implements them. Although it is a very simple and straightforward procedure if the code has been written with our patterns.

Associations in UML can have a great variety of features. The present work is limited to the analysis and implementation of multiplicity and navigability in binary associations. It excludes, therefore, more complex kinds of associations such as reflexive associations, whole/part associations (aggregations and compositions), qualified associations, association-classes, and n-ary associations [10]. It excludes, too, features such as ordering, changeability, etc.

The following sections of this article are devoted to studying the features of multiplicity, navigability and visibility of associations, with a detailed analysis of the possible problems and proposed solutions. Then, Section 3 contains the description of a uniform interface for all kinds of associations from the point of view of the participating classes, such as it is implemented by our patterns and source code. Finally, conclude briefly how to developed a concrete way of generating code of mapping UML associations using C# code in this works.

## 2 The Problem of Multiplicity

The multiplicity of a binary association, placed on an association end (the target end), specifies the number of target instances that may be associated with a single source instance across the given association, in other words, how many objects of one class (the target class) may be associated with a given single object from the other class (the source class) [2]. The classical example in Figure 1 illustrates binary multiplicity. Each instance of `Person` may work for none or one instance of `Company` (0..1), while each company may be linked to one or more persons (1..*). For those readers less familiarized with UML notation, the symbol (*) stands for "many" (unbounded number), and the ranges (1..1) and (0..*) may be abbreviated respectively as (1) and (*).


**Fig. 1.** A classical example of binary association with the expression of multiplicities

**Listing 1.** Program code to maintain the binary association between Person and Company

```
namespace model_1 {
  public class Company
  {
    ……
public Person[]person;
 } }
```

```
namespace model_1 {
   public class Person
   {
       ……
public Company company;
    }  }
```

The potential multiplicities in UML extend to any subset of nonnegative integers [2], not only a single interval as (2..*), or a comma-separated list of integer intervals as (1..3, 7..10, 15, 19..*): specifications of multiplicity like {prime numbers} or {squares of positive integers} are also valid, although there is no standard notation for them. Nevertheless, in UML as in other modeling techniques, the most usual multiplicities are (0..1), (1..1), (0..*) and (1..*). We are going to restrict our analysis to multiplicities that can be expressed as a single integer interval in the form of *(min..max)* notation. The multiplicity constraint is a kind of *invariant*, that is, a condition that must be satisfied by the system. A possible practice when programming is: do not check always the invariant, but only at the request of the programmer, after completing a set of operations that are supposed to leave system in a valid state (a *transaction*).

This practice is more efficient in run-time, and gives the programmer more freedom and responsibility in writing the code, with the corresponding risk that he or she forgets putting the necessary checks and carelessly leaves the system in a wrong state. On the other side, we think that checking multiplicity constraints is not very time consuming (inefficient), especially when compared with the time required to manage collections or synchronize bidirectional associations (see Section 3). Therefore, we think that it is worth doing as much as we can for the programmer, so that our first target will be to analyze the possibility of performing automatic checks for multiplicity constraints.

## 2.1 Optional and mandatory associations

The value of minimum multiplicity can be any positive integer, although the most common values are 0 or 1. When the value is 0 we say the association is *optional* for the class on the opposite end (class `Person` in Figure 1), when the value is 1 or greater we say it is *mandatory* (class `Company`). Optional associations pose no special problems for the implementation, but mandatory associations do. From a conceptual point of view, an object participating in a mandatory association needs to be linked *at any moment* with one object (or more) on the other side of the association, otherwise the system is in a wrong state. In the example given in Figure 1, an instance of `Company` needs always an instance of `Person`. Therefore, in the same moment you create the instance of `Company`, you have to link it to an instance of `Person`.

**This can happen in three different ways:**

- An instance of `Company` is created by an instance of `Person` and linked to its creator.
- An instance of `Company` is created with an instance of `Person` supplied as a parameter.
- An instance of `Company` is created and it issues the creation of an instance of `Person`.

The third case poses additional problems. The creation of a `Person` will probably require additional data, such as name, address, etc., and it does not seem very sensible to supply them in the creation of a `Company`. This problem becomes much worse if `Person` has other mandatory associations, for example one with the `Country` where he or she lives: if this were the case, the creation of a `Company` would require supplying data for creating a `Person`, for creating a `Country`, etc. The most obvious solution is to allow only the first and second forms of instantiation. But then suppose the association is mandatory in both ends. Which instance is to be created first? We have not a satisfactory choice, since we will put the system in a wrong state until both creations are finished.

We could think of an *atomic creation* of both instances, but this is valid only for the simplest case in which only two classes are involved. Should we define atomic creators for two, three, any number of classes? Similar problems arise when dealing with object deletion. Imagine now that we are not creating or deleting instances, but changing links between instances.

If you want to change the instance of `Company` that is linked with a given instance of `Person`, simply delete the link with the old `Company` and add a new link with the new `Company`. This works as far as the old `Company` is linked to other instances of `Person`; you can even delete the link and add no new one, since the association is optional for `Person`. If you had only one `Person` linked to a given `Company`, you should supply a new `Person` to the `Company` before deleting the link with the old `Person`, but this is only the specified behavior (the association is mandatory for `Company`) and you cannot complain about it. Nevertheless, we find new problems here. If the association with `Company` were mandatory for `Person` too

(that is, 1..1 multiplicity instead of the current 0..1), the instance of `Person` could not delete the old link with a `Company` and then add the new one, nor it could do it in the reverse order, "first add then delete", because it would go through a wrong system state. An *atomic change* of links would be valid only for the simplest cases, but not for more complex ones such as the following, rather twisted case (see Figure 2): consider classes `A` and `B`, which are associated with multiplicity 1..1 on both ends, and the corresponding instances `a1`, `a2`, `b1` and `b2`. In the initial state, we have the links `a1-b1` and `a2-b2`. In the final state, we want to have the links `a1-b2` and `a2-b1`. Even if we can change atomically `a1-b1` to `a1-b2` without violating the multiplicity constraints on `a1`, this would leave `b1` without any links and `b2` with two links until the final state is reached. We should have to perform the whole change atomically by means of an *atomic switch* implemented in a single operation.

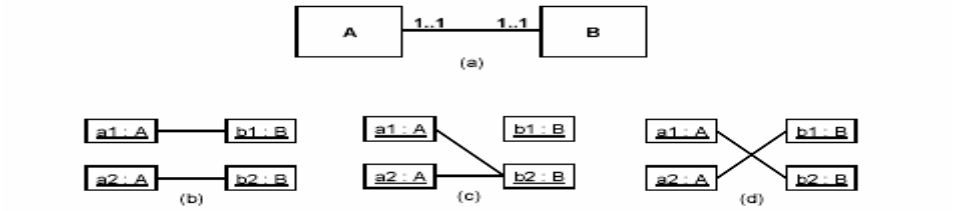



**Fig 2.** Multiplicity constraints can make very difficult changing links between instances without entering a wrong system state: a) class diagram; b) initial state; c) intermediate wrong state; d) final desired state.

Obviously, we cannot define a new operation to avoid any conceivable wrong state involving several instances. In consequence, we think that mandatory associations pose unsolvable problems regarding the creation and deletion of instances and links: we cannot achieve with a few primitive operations that a mandatory association is obeyed *at any time*, and we cannot isolate, inside atomic operations, the times when the constraint is not obeyed. Therefore, we have to relax the implications of mandatory associations for the implementation, as other methods do [8]. This proposal is as follows: *do not check the minimum multiplicity constraint when modifying the links of the association* (mutator methods, or setters), *but only when accessing them* (accessor methods, or getters). The programmer will be responsible for using the primitives in a consistent way so that a valid system state is reached as soon as possible.

For example, you will be allowed to create a `Company` without linking it to any `Person`, and you will be allowed to delete all the links of a `Company` with instances of `Person`; but before accessing, for other purposes, the links of that particular instance of `Company` towards any instances of `Person`, you will have to restore them to a valid state, otherwise you will get an *invalid multiplicity exception*, which shall be defined in the code that implements the associations according to this proposal [9,13].

### 2.2 Single and multiple associations

The value of maximum multiplicity in an association end can be any integer greater or equal than 1, although the most common values are 1 or *. When the value is 1 we say

the association is *single* for the class on the opposite end (class `Person` in Figure 1), when the value is 2 or greater we say it is *multiple* (class `Company`). Single associations are easier to implement than multiple associations: to store the only possible instance of a single association we usually employ an attribute having the corresponding target class as type, but to store the many potential links of a multiple association we must use some kind of collection of objects, such as the C# predefined `Length`, `Hashtable`, etc. In the general case we cannot use an array of objects, because it gets a fixed size when it is instantiated. Since collections in C# can have any number of elements, the maximum multiplicity constraint cannot be stated in the declaration of the collection in the C# code, but it must be checked elsewhere during run-time.

We need two kinds of mutators, `add` and `remove`, which will accept as a parameter either single objects or entire collections. Because of the problems with minimum multiplicity explained above, the remover sometimes will leave the source instance in a wrong state; we can't avoid this situation. The adder, instead, leaves us a wider choice. If we try to add some links above the maximum multiplicity constraint, we can choose between rejecting the addition or performing it; in the latter case we violate temporarily the constraint until a call to the remover restores the source instance to a safe state; the wrong state would only be detected by accessor methods, as we settled in the case of minimum multiplicity. However, this is true only for multiple associations implemented with a collection; in single associations implemented by means of an attribute we simply cannot violate the maximum multiplicity constraint: we are forced to reject the addition.

If we choose to reject the addition, instead, besides having an asymmetric behavior between remover and adder, we can find precedence problems when invoking the adder and the remover in succession. Consider class `Game` associated with class `Player` with multiplicity 2..4 (see Figure 3), and suppose an instance `g1` of `Game` is linked to two instances `p1`, `p2` of `Player`. We want to replace these two players by four new different players `q1`, `q2`, `q3`, `q4`. If we issue "first remove then add", we get finally what we want; if we issue "first add then remove", the addition is rejected and the remotion leaves the instance of `Game` in a wrong state.

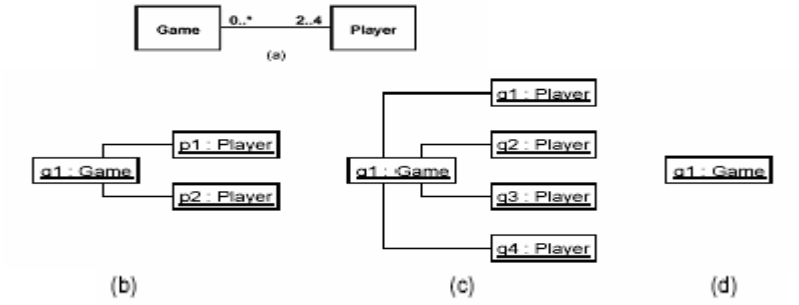



**Fig 3.** Precedence problems found when invoking the adder and the remover in succession: a) class diagram of `Game-Player` association; b) initial state with players `p1`, `p2`; c) final desired state after removing players `p1`, `p2` and then adding players `q1`, `q2`, `q3`, `q4`; d) final wrong state after unsuccessfully trying to add players `q1`, `q2`, `q3`, `q4` and then removing players `p1`, `p2`.

**Listing 2.** Program Code to maintain the binary association between Person and Company

```
namespace model_2 {

public class Game {

public Player m_Player;

public Game(){

          }

~Game(){

          }

public virtual void
Dispose(){

}   }//end Game }//end
namespace model_2
```

```
namespace model_2  {

public class Player {

public Player(){

          }

          ~Player(){

          }

public virtual void
Dispose(){

          }

}//end Player }//end
namespace model_2
```

In the end, we have preferred to *reject the addition if it violates the maximum allowed*, and ask the users of mutator methods to use them always in the right order, first remove then add, so that we can get an analogous behavior for single and multiple associations. Therefore, the remover does not check the minimum multiplicity constraint (possibly leaving empty a mandatory association), the adder does check the maximum multiplicity constraint, and the getter raises an exception if either constraint is not fulfilled. Accessor methods of multiple associations have another peculiarity, when compared with the accessors of single associations: they return a collection of objects, not a single object, therefore the returned type is that of the collection, not that of the target class. In our implementation, the returned type is the C# interface `Collection`, which is implemented by all standard collections. Internally, we use a `Hashtable` collection, which ensures that there are no duplicate links in an association, as the UML requires [7].

Finally, the standard collections in C# are specified to contain instances of the standard class `Object`, which is a superclass of every class in C#. You cannot specialize these collections to store objects pertaining only to a particular class (That is, you cannot specialize them to modify their storage structure, but you can modify their behavior so that they) store in effect only the required objects, precisely by means of the run-time type checking method we describe.. This means that, if we use a `Hashtable` inside `Company` to store the links to instances of `Person`, we must ensure on our own that no one puts a link to an instance of another class such as `Dog` or `Report` (this could happen if a collection of objects is passed as a parameter to the `add` method). Therefore, the mutator methods must perform a run-time type checking by means of explicit *casting*. If the type-check fails, then the link is not set to that object, and a *class cast exception*, which is predefined in C#, is raised.

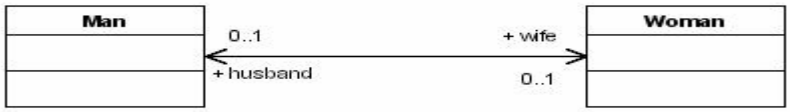
## 3 The Problem of Navigability

The directionality, or navigability, of a binary association, graphically expressed as an open arrow at the end of the association line that connects two classes, specifies the ability of an instance of the source class to access the instances of the target class by means of the association instances (links) that connect them (An alternate definition: the possibility for a source object to designate a target object through an association instance (link), in order to manipulate or access it in an interaction with message interchanges. The Standard does not give a clear definition of navigability, as we have shown in previous works where we have tried to clarify this topic [9,10,13]). In this paper, we take navigability and directionality as synonyms. If the association can be traversed in both directions, then it is *bidirectional* (two-way), otherwise it is *unidirectional* (one-way).

A navigable association end, which is referenced by its rolename, defines a pseudo attribute of the source class, so that the source instance can use the rolename in expressions in the same way as it uses its own attributes [6]. An instance can communicate (by sending messages) with the connected instances of the opposite navigable end [11], and it can use references to them as arguments or reply values in communications [7]. Similarly, if the association end is navigable, the source instance can query and update the links that connect it to the target instances.

The examples in Figure 4 illustrate navigability. The association `Key-Door` is unidirectional, meaning that a `Key` can access the `Door` it can open, but an instance of `Door` does not know the set of instances of `Key` that can open it: the `Door` cannot traverse the connections (links) against the navigability of the association. On the other side, the association `Man-Woman` is bidirectional, meaning that connected instances of these classes know each other.


a) Unidirectional




b) Bidirectional

**Fig 4.** Examples of a) Unidirectional and b) Bidirectional Associations.

The arrowheads can be shown or omitted in a bidirectional association [14]. Unfortunately, this leads to an ambiguity in the graphical notation, because we cannot. distinguish between bidirectional associations and associations with unspecified navigability. Or, worse, unspecified associations are assumed to be bidirectional without further analysis [10].

**Listing 3.** Program code to maintain the unidirectional and bidirectional associations

```
namespace model_3 {
  public class Key_ {
  ……
  public Door door;
        }     }
  public class Door {
    ……
       }
```

```
namespace model_4 {
   public class Woman {
    ……
   public Man husband;
   }     }
   public class Man {
    ……
  public Woman woman; }
```

### 3.1 Unidirectional associations

A *single unidirectional association* is very similar to a single valued attribute in the source class, of the type of the target class: an embedded reference, pointer, or whatever you want to call it. The equivalence, however, is not complete. Whereas the *attribute value* is "owned" by the class instance and has no identity, an *external referenced object* has identity and can be shared by instances of other classes that have a reference to the same object [12] (see Figure 5). Anyhow, the equivalence is satisfactory enough to serve as a basis for the implementation of this kind of associations. In fact, in C# there is no difference at all: except for the case of primitive values, attributes in C# are objects with identity, and if they are public you cannot avoid them to be referenced and shared by other objects.

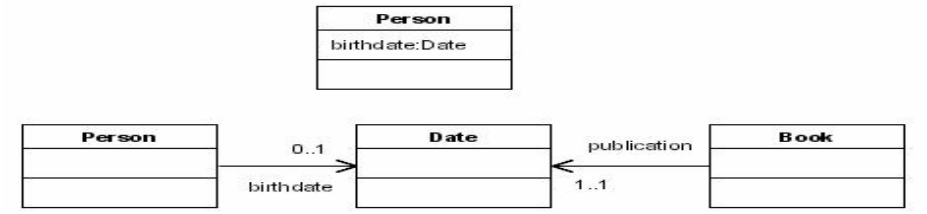



**Fig. 5.** Partial equivalence between a) attribute and b) single unidirectional association.

**Listing 4.** Program code to maintain the Partial equivalence between attribute and single unidirectional association

```
namespace model_2 {
 public class Person {
```

```
……..
public Date birthdate;
```

```
            }                          .....

            }                          public Date publica-
                                       tion;
public class Date {
                                                  }
/// Attributes - Asso-
ciation End                                       }

public class Book {
```

A *multiple unidirectional association* is a bit more complicated, although the implementation can be based on the same principles, since it can be assimilated to a multivalued attribute (UML allows multiplicity in attributes, thus multivalued attributes [8]). To manage the collection of objects on the navigable end, however, we need an additional object of a standard collection class, which is a `Hashtable` in our implementation (see Figure 6).

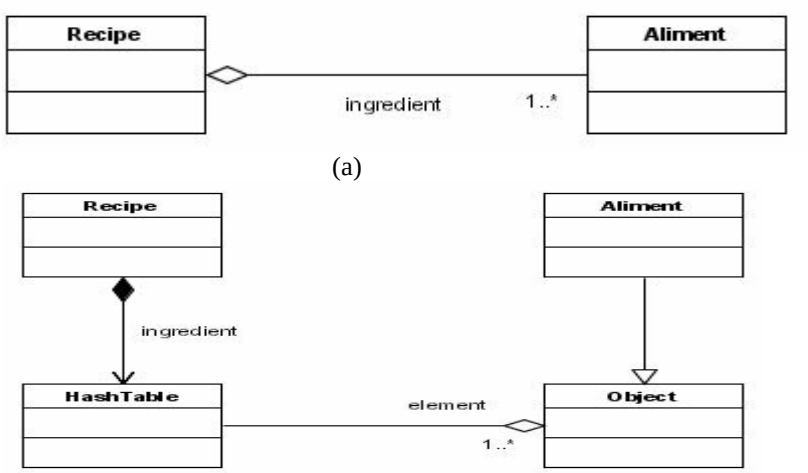



**Fig. 6.** Multiple unidirectional association: a) analysis diagram and b) design diagram.

A new object must be inserted to manage the collection of target objects. The standard collections in C#, such as `Hashtable`, are defined for the standard class `Object`, which is a superclass of every class; therefore, mutator methods must ensure that the objects contained in the collection parameter are of the appropriate type before adding them to the collection attribute. Therefore, the type of the attribute used to implement the association inside the source class is not any more the target class itself, but the `Hashtable` class or another convenient collection class. The methods to manage the association will have to accomplish some additional tasks. *Mutators* can add or remove not only single objects of the class target, but also entire collections; thus, the type of the parameter will be either the target class of the association or the intermediate collection class.

In this case, mutator methods must ensure that the objects contained in the collection parameter are of the appropriate type before adding them to the collection attribute. *Accessors*, as we have already explained (see Section 2), do not return a single object, but a collection of objects, even when the collection is made up of only one element. The returned collection object is not identically the same one that is stored inside the source class, but a clone (a new object with a collection of references to the same target elements), because the original collection object must remain completely encapsulated inside the source object (represented by the composition in Figure 6).

**Listing 6.** Program code to maintain the Multiple unidirectional association analysis and design diagrams

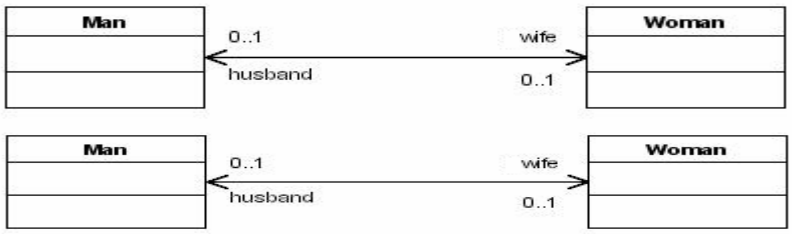
```
namespace model_6  {

   public class Recipe
          {

    ……..

    public Aliment[]
ingredient;

    ……..

   public Aliment ali-
ment;

          }   }

public class Aliment {

    ……..

public Recipe recipe;

          }

public class Recipe {

   ……..

public HashTable
hashTable;

public HashTable
hashTable_1; }

public class Aliment :
Object {

          }

public class HashTable
{

    ……..

public Recipe recipe;

    ……..

public Object[] object;

           }

public class Object {

    ……..

public HashTable
hashTable_2;

          }
```

As the diagrams in Figures 5 and 6 show, in our opinion *the multiplicity constraint in a design model can be specified only for a navigable association end*. Indeed, the multiplicity is a constraint that must be evaluated within the context of the class that owns the association end; if that class knows the constraint, then it knows the association end, that is, the end is navigable. You cannot restrict the number of objects connected to a given instance unless this instance has some knowledge of the connected

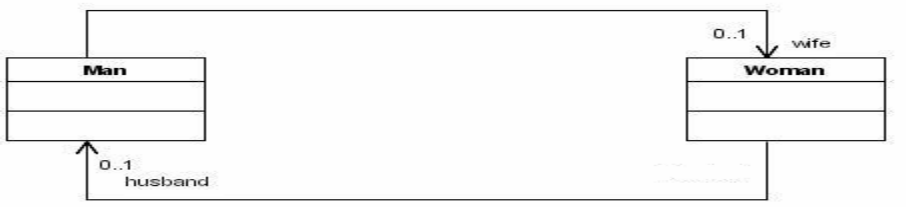
objects, that is, unless you make the association end navigable. Therefore, *the need for a multiplicity constraint other than 0..** (that is, unrestricted) *is an indication that the association end must be navigable*. In consequence, unidirectional associations with multiplicity constraints on the nonnavigable association end must be rejected in code generation.

### 3.2 Bidirectional associations

The partial equivalence between attributes and unidirectional associations is not any more found among bidirectional associations. Instead, an instance of a bidirectional association is more like a *tuple of elements* [14]. Combining the multiplicities in both association ends, we can have three cases: single-single, single-multiple, and multiple_multiple.




**a) analysis diagram**




**b) design diagram**.

**Fig. 7.** Single-single bidirectional association: a) analysis diagram and b) design diagram

**Listing 7.** Program code to maintain the Single-single bidirectional association analysis and design diagrams

```
namespace model_7 {

public class Man {

    ……

public Woman woman;    }    }

public class Woman {
```

```
/// Attributes - AssociationEnd and AssociationEnd hus-
band

public Man husband;  } }
```

The implementation of the association's mutators must ensure that the husband of the wife of a given man is that man himself, and vice versa.An easy way to implement a *single-single* bidirectional association is by means of two synchronized single unidirectional associations (see Figure 7). The synchronization of the two halves must be preserved by the mutator methods on each side: every time an update is requested on one side, the other side must be informed to perform the corresponding update; the update is accomplished only if both sides agree that they can perform it while keeping maximum multiplicity constraints.

A *single-multiple* bidirectional association can be implemented in a similar way, combining a single unidirectional association and a multiple unidirectional association. And, finally, a *multiple-multiple* bidirectional association is achieved by means of two multiple unidirectional associations (see Figure 8).

Synchronization becomes progressively a more and more complex issue when one or both association ends are multiple. Consider the example given in Figure 8. Suppose you want to add an author to a particular `Book` instance; you do this by issuing the `add` method on the `Book` instance, and passing a `Person` instance as a parameter. If the `Book` can have more authors without violating its maximum multiplicity (which is 3), then it requests the author to add the `Book` itself to the collection of publications the `Person` has; this can fail if the maximum multiplicity constraint for the number of publications (in this case, 10) is violated. If the request to the author succeeds, then the `Book` updates its side.

Now, you can try adding a collection of authors to a `Book`, too. As one can expect, the `Book` requests each one of the authors to add the `Book` itself as a publication; if only one of the authors fails to add the `Book`, then the whole operation must be undone, since an update must be atomic: all or none.

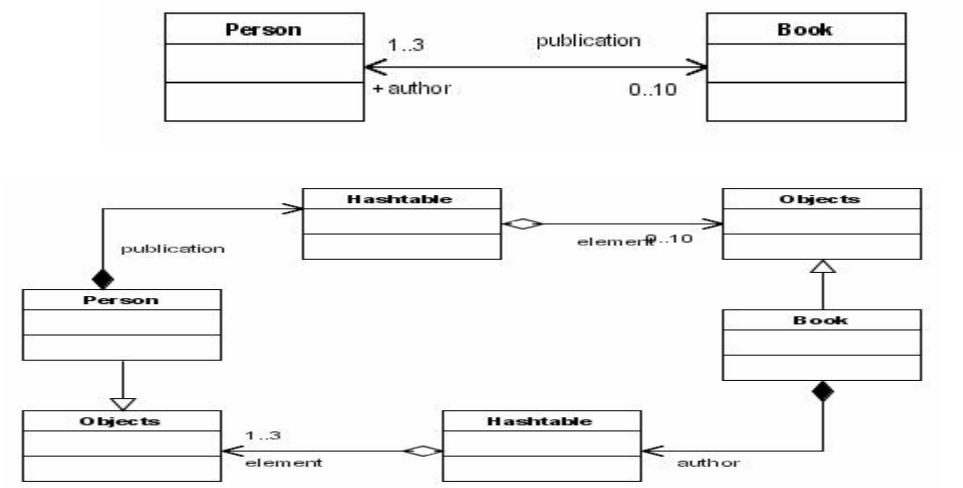



**Fig. 8.** Multiple-multiple bidirectional association: analysis and design diagrams

Similar considerations apply to the `remove` mutator, bearing in mind that the `remove` method is performed even if the minimum multiplicity constraint is not kept, therefore it can leave the source instance or any of the affected target instances in an invalid state.

## 4 Conclusions

In this work we have developed a concrete way of generating code of mapping UML associations using C# code: we have written specific code patterns, and we have using a tool that reads a UML design model stored in XMI format and generates the necessary C# files. We have paid special attention to two main features of associations: multiplicity and navigability. This analysis has encountered difficulties that may reveal some weaknesses of the UML Specification.

However, different tool options will allow the user to override the automatic multiplicity and type checks when generating code, in favor of efficiency. Besides, we have argued that unidirectional associations should not have a multiplicity constraint on the source end in a design model, and bidirectional associations should not have both ends with private (or protected) visibility; therefore, the tool will reject the generation of code for these associations. Again, the user will be able to disable this model-correctness checking and issue the code generation at his/her own risk.

This work can be continued on several lines. First, implementation of other association end properties, such as ordering, changeability, interface specified, xored associations, and so on. Second, specific implementation of particular kinds of binary associations, such as reflexive associations, aggregations and compositions. Third, implementation of more complex associations: qualified associations, associations classes, and n-ary associations. Fourth, expand the tool to perform reverse engineering, that is, obtaining the associations between classes by analyzing the code that implements them.

## References

1. Object Management Group. *XML Metadata Interchange (XMI) Specification*, Version 1.2, January 2002. Available at http://www.omg.org/.
2. James Rumbaugh. "A Search for Values: Attributes and Associations". *Journal of Object Oriented Programming*, 9(3):6-8, June 1996.
3. Scott W. Ambler. “An Overview of Object Relationships”, “Unidirectional Object Relationships”, “Implementing One-to-Many Object Relationships”, “Implementing Many-to-Many Object Relationships”. A series of tips to be found at IBM Developer Works, http://www-106.ibm.com/developerworks/.
4. Hermann Kaindl. “Difficulties in the Transition from OO Analysis to Design”. *IEEE Software*, 16(5):94-102 (1999).
5. The Poseidon UML tool, http://www.gentleware.com//
6. James Rumbaugh. "Relations as Semantic Constructs in an Object- Oriented Language", In *Proceedings of the ACM Conference on Object-Oriented Programming: Systems, Languages and Applications*, pp. 466-481, Orlando, Florida, 1987.

7. Object Management Group. *Unified Modeling Language (UML) Specification*, Version 1.4, September 2001 (Version 1.3, June 1999). Available at http://www.omg.org/.
8. Perdita Stevens. “On the Interpretation of Binary Associations in the Unified Modelling Language”, *Journal on Software and Systems Modeling*, 1(1):68-79 (2002). A preliminar version in: Perdita Stevens. "On Associations in the Unified Modeling Language". *The Fourth International Conference on the Unified Modeling Language*, UML'2001, October 1-5, 2001, Toronto, Ontario, Canada. Published in *Lecture Notesin Computer Science 285*, Springer 2001, pp. 361-375.
9. Gonzalo Génova. "Semantics of Navigability in UML Associations". *Technical Report UC3M-TR-CS-2001-06*, Computer Science Department, Carlos III University of Madrid, November 2001, pp. 233-251.
10. Gonzalo Génova, Juan Llorens, Paloma Martínez. “The Meaning of Multiplicity of N-ary Associations in UML”, *Software and Systems Modeling*, 1(2): 86-97, 2002. A preliminary version in: Gonzalo Génova, Juan Llorens, Paloma Martínez. “Semantics of the Minimum Multiplicity in Ternary Associations in UML”. *The 4th International Conference on the Unified Modeling Language-UML'2001*, October 1-5 2001, Toronto, Ontario, Canada. Published in *Lecture Notes in Computer Science 285*, Springer 2001, pp. 329-341.
11. Gonzalo Génova, Juan Llorens, Vicente Palacios. "Sending Messages in UML", *Journal of Object Technology*, vol.2, no.1, Jan-Feb 2003, pp. 99- 115, http://www.jot.fm/issues/issue_2003_01/article3.
12. Il-Yeol Song, Mary Evans, E.K. Park. "A Comparative Analysis of Entity- Relationship Diagrams", *Journal of Computer and Software Engineering*, 3(4):427-459 (1995).
13. William Harrison, Charles Barton, Mukund Raghavachari. "Mapping UML Designs to Java". *The 15th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications-OOPSLA’2000*, October 15-19 2000, Minneapolis, Minnesota, United States. *ACM SIGPLAN Notices*, 35(10): 178-187. ACM Press, New York, NY, USA.
14. Guy Genilloud. "Informal UML 1.3 - Remarks, Questions, and some Answers". *UML Semantics FAQ Workshop* (held at ECOOP'99), Lisbon, Portugal, June 12th 1999.
15. *The Fujaba CASE Tool*, University of Paderborn, http://www.fujaba.de/.
16. Java Community Process. *Java Metadata Interface (JMI) Specification*, Version 1.0, June 2002. Available at http://www.jcp.org/.

# Architecture for an Autonomic Web Services Environment

Wenhu Tian, Farhana Zulkernine, Jared Zebedee, Wendy Powley and Pat Martin
School of Computing,

Queen's University, Kingston, ON Canada
{tian, farhana, zebedee, wendy, martin }@cs.queensu.ca

**Abstract.** The growing complexity of Web service platforms and their dynamically varying workloads make manually managing their performance a tough and time consuming task. Autonomic computing systems, that is, systems that are self-configuring and self-managing, have emerged as a promising approach to dealing with this increasing complexity. In this paper we propose an architecture of an autonomic Web service environment based on reflective programming techniques, where components at a Web service hosting site tunes themselves and collaborate to provide a self-managed and self-optimized system.

## 1 Introduction

Web services are self-contained and self-describing software components that can be accessed over the Internet. They are now well accepted in Enterprise Application Integration (EAI) [19] and Business to Business Integration (B2Bi) [4]. Performance plays a crucial role in promoting the acceptance and widespread usage of Web services. Poor performance (e.g. long response time) means the loss of customers and revenue [14]. In the presence of a Service Level Agreement (SLA), failing to meet performance objectives could result in serious financial penalties for the service providers. As a result, Web service performance is of utmost importance, and recently has gained a considerable amount of attention [3, 15, 18].

A Web service is a Web-accessible program that is described in a WSDL (Web Service Description Language) [17] document. Web services are published or discovered via a UDDI (Universal Description, Discovery and Integration) [16] registry. SOAP (Simple Object Access Protocol) [13] is the most common message passing protocol used to communicate with Web services.

A Web service hosting site typically consists of many individual components such as HTTP servers, application servers, Web service applications, and supporting software such as database management systems. If any component is not properly configured or tuned, the overall performance of the Web service suffers. For example, if the application server is not configured with enough working threads, the system can perform poorly when the workload surges. Typically components such as HTTP servers, application servers or database servers are manually configured, and manually tuned. To dynamically adjust in an ever-changing environment, these tasks must be automated.

Unacceptable Web service performance results from both networking and server-side issues [10]. Most often the cause is congested applications and data servers at the service provider's site as these servers are poorly configured and tuned. Expert administrators, knowledgeable in areas such as workload identification, system modeling, capacity planning, and system tuning, are required to ensure high performance in a Web service environment. However, these administrators face increasingly more difficult challenges brought by the growing functionalities and complexities of Web service systems, which stems from several sources:

- **Increased emphasis on Quality of Services**

  Web services are beginning to provide Quality of Service features. They must guarantee their service level in order that the overall business process goals can be successfully achieved.

- **Advances in functionality, connectivity, availability and heterogeneity**

  Advanced functions such as logging, security, compression, caching, and so on are an integral part of Web service systems. Efficient management and use of these functionalities require a high level of expertise. Additionally, Web services are incorporating many existing heterogeneous applications such as JavaBeans, database systems, CORBA-based applications, or Message Queuing software, which further complicate performance tuning.

- **Workload diversity and variability**

  Dynamic business environments that incorporate Web services bring a broad diversity of workloads in terms of type and intensity. Web service systems must be capable of handling the varying workloads.

- **Multi-tier architecture**

  A typical Web service architecture is multi-tiered. Each tier is a sub-system, which requires different tuning expertise. The dependencies among these tiers are also factors to consider when tuning individual sub-systems.

- **Service dependency**

  A Web service that integrates with external services becomes dependent upon them. Poor performance of an external service can have a negative impact on the Web service.

Autonomic Computing [7] has emerged as a solution for dealing with the increasing complexity of managing and tuning computing environments. Computing systems that feature the following four characteristics are referred to as Autonomic Systems:

- **Self-configuring -** Define themselves on-the fly to adapt to a dynamically changing environment.
- **Self-healing -** Identify and fix the failed components without introducing apparent disruption.

- **Self-optimizing -** Achieve optimal performance by self-monitoring and self-tuning resources.
- **Self-protecting -** Protect themselves from attacks by managing user access, detecting intrusions and providing recovery capabilities.

In this paper we propose an architecture for an autonomic Web services environment. We consider each component in the proposed architecture as self-managing and thereby present a hierarchical layout of autonomic managers that constitute a self-configuring and self-optimizing autonomic Web service system. The remainder of the paper is structured as follows. Section 2 discusses related approaches to Web service management. Our proposed autonomic architecture is presented in Section 3, and a detailed scenario to illustrate how the architecture works is provided in Section 4. Section 5 summarizes and concludes the paper.

## 2 Related Work

Architectural approaches based on SLA-driven Web services have been proposed by Dan et al. [5] and Levy et al. [9]. Dan's framework includes components for the support of an SLA throughout its entire life-cycle as well as SLA-driven management of services. Levy et al uses a queuing model to predict response times for different resource allocations. In their model, the management system is transparent and allocates server resources dynamically to maximize the expected value of a given cluster utility function. Both of these approaches focus on service provisioning. We focus on autonomic management rather than the provisioning aspects.

Farrell and Kreger [6] propose a number of principles for the management of Web services including the separation of the management interface from the business interface, pushing core metric collection down to the Web services infrastructure. They use intermediate Web services that act as event collectors and managers. We incorporate these ideas and expand upon them in our approach.

The insufficient reliability and lack of autonomic features in current Web services architectures is presented by Birman et al in [2]. He proposes some extensions to the current Web services framework in the form of more robust monitoring and reliable messaging to achieve higher availability.

## 3 Autonomic Web Services Architecture

A Web services environment typically consists of a collection of components including HTTP servers, application servers, database servers, and Web service applications. In our proposed architecture, as shown in Figure 1, we consider each component to be autonomic, that is, self-aware and capable of self-configuration to maintain a specified level of performance. System-wide management of the Web services environment is facilitated by a hierarchy of Autonomic Managers that query other managers at the lower level to acquire current and past performance statistics, consolidate the data from various sources, and use pre-defined policies and SLAs to assist in system-wide tuning.

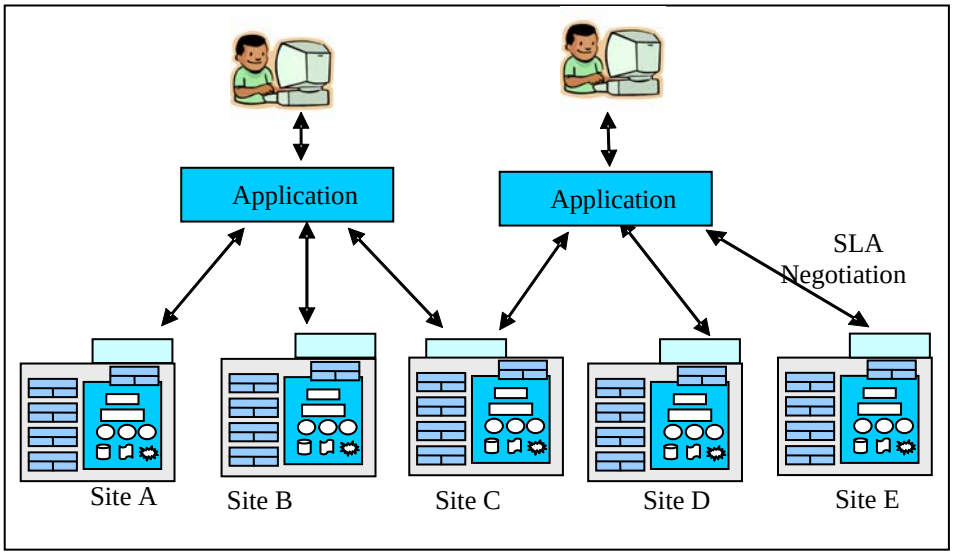



**Fig. 1.** Autonomic Web Services Architecture

At the lowest level in our architectural hierarchy are the Au*tonomic Elements*. We refer to an *autonomic element* as a component augmented with self-managing capabilities. An autonomic element is capable of monitoring the performance of its component, or *managed element*, (such as a DBMS or an HTTP server), analyzing its performance and, if required, proposing and implementing a plan for reconfiguration of the managed element. Autonomic elements form the building blocks of our architecture and are described in more detail in Section 3.1.

We refer to a *Site* as a collection of components and resources necessary for hosting a Web service system provided by an organization. A Web services hosting site typically consists of HTTP servers, application servers, SOAP Engines, and Web services. Web services are basically Web accessible interfaces or applications that can connect to other backend applications such as legacy systems, or database management systems. Most often these backend components are located on separate servers that are connected by a Local Area Network (LAN). A site can therefore span multiple servers. A *site manager* oversees the overall performance of the site and provides service provisioning for the components associated with the site.

An *Application,* as shown in Figure 1, is a special purpose client program that uses one or more Web services, possibly from different sites. An investor application, for example, that allows users to look up stock prices may use Web services from several different companies. A site's *SLA Negotiator* negotiates SLA agreements between the applications and the Web services hosted by the site. Once SLA agreements are made, the site must manage its resources to ensure the agreed level of performance.

There are two levels of management in our approach; the component level and the site level. The component is responsible for managing its own performance to meet goals specified by the site manager. The site manager monitors for SLA compliance, sets component goals, and provides resource provisioning when necessary.

### 3.1 Autonomic Elements

An autonomic element can be viewed as a feedback control loop as shown in Figure 2 [8], controlled by an *Autonomic Manager*. The autonomic manager oversees the monitoring of the component (the *Managed Element*), and by analyzing the collected statistics in light of known policies and goals, it determines whether or not the component performance is adequate. If necessary, a plan for reconfiguration is generated and executed.

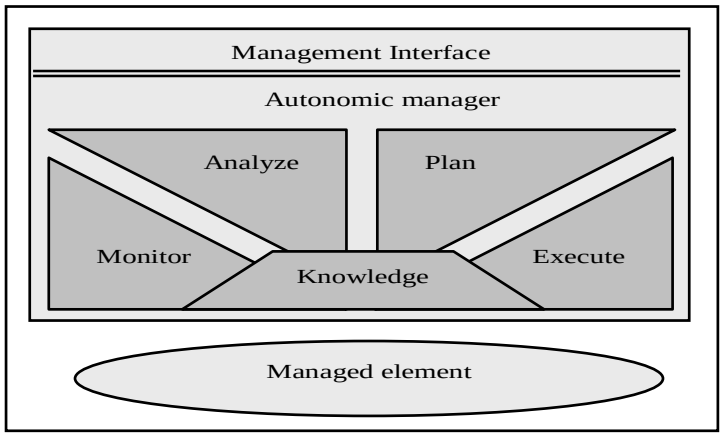



**Fig. 2.** Autonomic Element

One approach to building autonomic elements is based on the principles of *reflective programming* [11]. A reflective system is one that can inspect and adapt its internal behaviour in response to changing conditions. Typically a reflective system maintains a model of self-representation, and changes to the self-representation are automatically reflected in the underlying system.

An example of an autonomic database management system (DBMS) based on reflective programming techniques, was presented by Martin et al [12]. In this system, the self-representation of the system embodies the current configuration settings and the statistics that are collected regarding the system performance. This information is stored as a set of database relations that can be queried and updated. A monitoring tool periodically takes snapshots of the DBMS performance and stores the collected data in a data warehouse. When a new set of performance data is inserted into the data warehouse, a database trigger is fired that calls a diagnosis function. The diagnosis function compares current and past performance data to determine whether or not a change in configuration is warranted based on a preset desired performance setting. If one or more configuration parameters should be altered, a change is made to the self-representation which in turn triggers a change to the underlying DBMS configuration parameters.

We use this notion of reflection to implement Web components as autonomic elements. In our architecture, all components such as the HTTP server, the application server, the Web services and supporting applications as well as the site manager are instances of autonomic elements. Each component has an autonomic manager as shown in Figure 2, augmented with a reflective *Management Interface*. This interface

is used by higher level managers to set performance goals as per Service Level Agreements (SLAs) for the managed element and to obtain current performance statistics for the component. As in the example of the autonomic DBMS, a monitoring tool periodically monitors the system performance and the analyzer compares the current and past performance to determine whether a configuration change is necessary to achieve the desired goal. Following the principles of reflective systems, each autonomic element maintains a self-representation which embodies the component's current goal settings and its current performance statistics. Updates made to the self-representation trigger changes to the actual system. If deemed necessary by the analyzer, changes are made to the self-representation to reconfigure the component.

In our proposed architecture, to ensure interoperability between autonomic elements, a common management interface is specified for all elements to provide access to the self-representation. Each autonomic element monitors itself to assess its general health and the performance data is stored as part of the component's self-representation. This data can be accessed using the methods provided by the management interface. Historical data may be used for performance analysis and prediction.

The standard Web services environment already provides the tools required to define, publish, discover, and to use APIs across platforms. These tools and methods are exploited in our proposed architecture for communication between elements. To implement the reflective interface, we view each component as a Web service where the self-representation is accessed via Web service operations for each element. Two management interfaces are defined for each autonomic element; the *Performance Interface* and the *Goal Interface*. The Performance Interface exposes methods to retrieve, query and update performance data. Each element exposes the same set of methods, but the actual data each provides varies. Meta-data methods allow the discovery of the type of data that is stored for each element

```
public interface Goal{
      // retrieves a list of goals that can be set for the component

      public Vector getMetaData();
      // retrieves the current goal for the component
      public Double getGoal (String goalType);
      // set a goal for the component
      public Boolean setGoal(String goalType, Double
value)
}

public interface Performance{
      // retrieves a list of goals that can be set for the component
      public Vector getMetaData();
      // retrieves the most recent performance data
      public Vector getCurrentData();
      // returns a specified portion of the most recent performance
```

**Fig. 3.** Management Interface Specifications

The Goal Interface provides methods to query and establish the goals for an autonomic element. Meta-data methods promote the discovery of associated goals and additional methods allow the retrieval of current goals. Goals for individual components can be set only by their associated site manager. Goals for a site manager are set by the site's SLA Negotiator component.

Component-level performance interfaces are accessed only by their associated site manager. A site manager uses the performance interface to asses the current health of each of its components and uses the component's goal interface to set individual goals for each component.

Management interfaces are defined and published using WSDL and a private management UDDI registry as suggested by Farrell and Kreger [6]. The self-representation can be stored using any storage format (database, log files etc) as these details are made transparent by the use of a Web service interface. Figure 3 shows the interface specification of the management interfaces common to all autonomic elements. The WSDL specification for the *setGoal()* method is given in the Appendix as an example.

Each autonomic element implements a monitoring component to asses the health of its managed element. Monitoring incurs a certain degree of overhead, so monitoring processes must be lightweight and invoked as infrequently as possible. Multiple levels of monitoring allow more information to be collected depending on the amount of detail that is desired. In some cases, it may be desirable to *drill down*, collecting more detailed information to assist in problem determination. At times of stable, acceptable performance, it may suffice to collect data less frequently.

Current HTTP servers and application servers provide rich interfaces for monitoring tools to extract performance statistics and running status. A variety of monitor tools are available on the market to visualize and analyze collected statistics, and if necessary, to fire warnings when the pre-set thresholds are violated [20, 1]. DBMSs are rich in monitoring tools and APIs for gathering information. Monitors can be switched on or off at will, and different levels of monitoring can be specified. Monitoring individual Web services presents more of a challenge as each Web service application is unique. Generic monitors can be developed that provide basic information such as response time for the Web service, number of requests per time unit, or average queue length.

### 3.2 Site Management

A *site* is a collection of Web service components and resources provided by an organization that offers one or more Web services. The components comprising a site are shown in Figure 4. A site may be distributed across many physical nodes. Multiple instances of a component may reside on the same site and resources are provisioned as required.

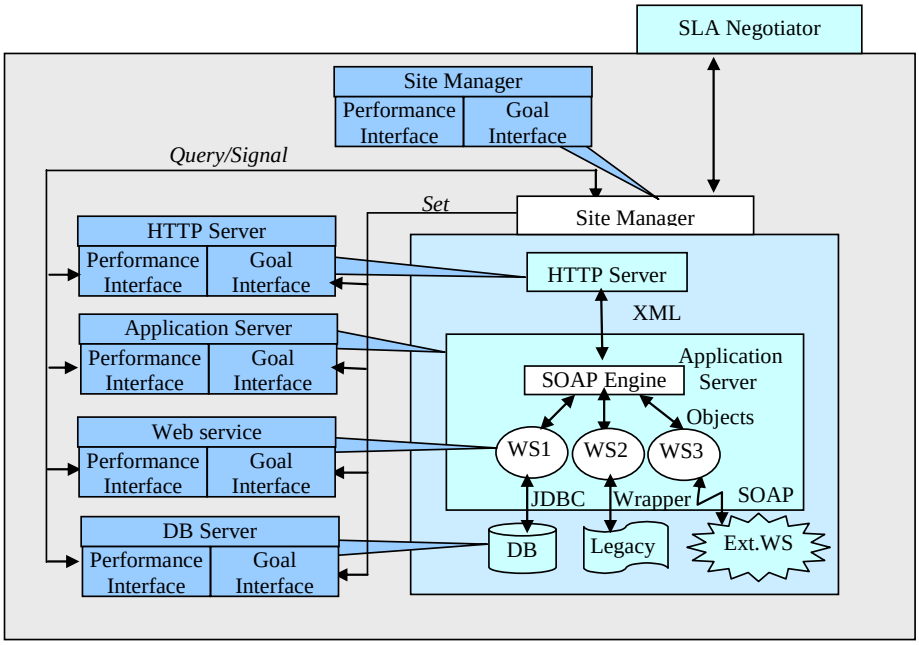



**Fig. 4.** Autonomic Web Services Site

Applications that wish to use the Web services offered by a site negotiate a SLA with the site's SLA Negotiator. Details of an automated approach to SLA negotiation is presented by Dan et al in [5], and is beyond the scope of this paper. We assume that different SLAs can be specified for each Web service or, if a finer level of granularity is required, SLAs can be set on a per-operation level. The site's SLA Negotiator translates these high level specifications into performance goals such as response time or average throughput for each Web service or operation. The SLA Negotiator component sets the goals for the site using the site's management interface.

Each site employs a *Site Manager* that oversees the general performance of the components comprising the site. The site manager itself is implemented as an autonomic element with its own autonomic manager. Conceptually, the site manager is the autonomic manager of all the components within the scope of the site. The site manager collects the performance statistics of each component by querying the management interfaces of the individual components. This information, along with the policies and goals defined for the site, is used to determine whether or not the performance of the site is adequate. If the site is in violation of one or more of the SLA agreements, an action plan is generated and executed. An action plan may involve the generation and setting of new goals for particular components, or it may involve a modification in the provisioning of resources.

The site manager is implemented as a Web service that exposes the site's performance interface that can be accessed by other site managers or external components. This interface can be used by applications for error tracking, Web service selection, or by modules handling external SLA compliance monitoring. The

performance data for a site provides summary data indicating the overall performance of the associated components.

The site manager is responsible for monitoring the overall performance of the Web services offered by the site. The site manager retrieves the performance data via the components' performance interfaces. The information required by the component for self-management may differ from that required for overall system management by managers at the site level. For instance, a DBMS focuses on low level resources such as I/O and CPU usage to maximize performance. To optimize site performance, and to monitor SLA compliance, the site manger requires higher level statistics such as throughput or transaction response times. This information is available through the components management interface.

## 4 Scenario

Functionality of the different components presented in the architecture of autonomic Web services system can be better explained using a common example like the Stock Quote composite Web service system shown in Fig. 5. In this system, a customer uses an *Investor* application to find out the details about multiple stocks. The *Investor* application invokes a *Stock Broker (SB)* Web service by sending a *register* message containing a list of stock IDs. The Stock Broker sends *accept* or *reject* message to the Investor in response. In case of *accept*, the Stock Broker sends the stock IDs received from the customer, one by one to the *Research Department (RD)* Web service. The RD finds the necessary information and sends a *report* directly to the Investor application. When the Investor receives information about all the stocks, it sends an *acknowledgement* message to the Stock Broker service. The Stock Broker service then submits the *bill* to the Investor and notifies the Research Department about the end of the job. The messages interchanged in this system are presented in Figure 5.

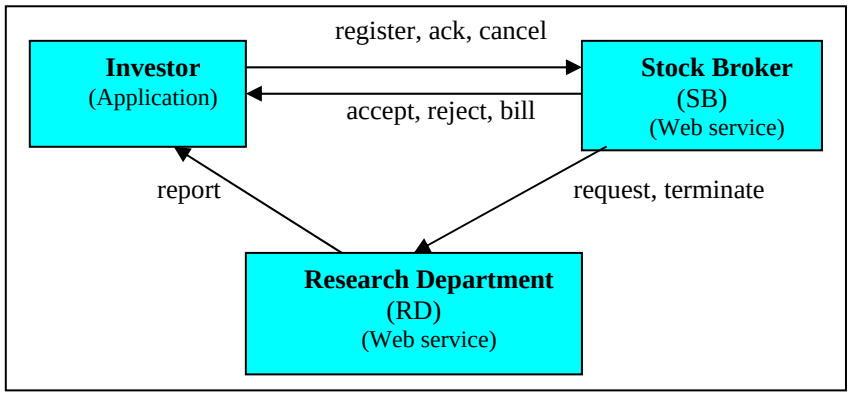



**Fig. 5.** Stock Broker Web Service System

The Stock Broker and Research Department Web services are located at two different sites. Each of these sites is managed by a site manager. The site manager receives the SLA from the SLA negotiator and monitors the performance of the

different components at the site to provide an overall performance in compliance with the SLA. For the Stock Broker service system, the site manager monitors the performances of the HTTP server, application server, and other components at the site including the Stock Broker service.

If the SLA between the Investor and the Stock Broker site is in violation, the Stock Broker's site manager retrieves the performance data of all the individual components associated with this site, analyzes them, and sets new goals for the necessary components in order to avoid violation of the SLA. For example, if the maximum response time specified in the SLA is five seconds, and the observed response time is close to, or beyond this threshold, the site manager tries to set new goals for specific components to reduce the response time to five seconds or less. If the perceived bottleneck is the HTTP server, the site manager uses the HTTP server's goal interface to set a new goal for this component.

Each component in the autonomic Web service system is associated with its own autonomic manager. When new performance goals are set, the specific components attempt to reconfigure themselves using their own autonomic managers. In our example, the HTTP server's autonomic manager may increase the number of threads to improve its response time.

At the highest level, the client Investor application sets the SLA for the Stock Broker service through the SLA negotiator before invoking the service. The SLA negotiator conveys the same to the Stock Broker's site manager and also to the linked services, in this case the Research Department. When all the linked services agree to the SLA, the Investor application can invoke the Stock Broker service. Both the application and the site manager monitor the service performance to ensure SLA compliance. For linked services, the site manager of the calling service does the monitoring while the SLA negotiator plays the role of the application in doing the SLA negotiation with the linked services.

## 5 Summary

Performance plays a crucial role in the eventual acceptance and widespread adoption of the Web services model of application deployment. Web service performance, however, is difficult to manage because of the complexity of the components and their interactions, and the variability in demand and the environment. In this paper, we propose autonomic computing as a solution to the problems in managing Web service performance. We describe an architecture for an autonomic Web services environment where each component is fully autonomic and equipped to cooperate in a managed environment. Each component provides a management interface that exposes a self-representation consisting of performance statistics and goal information. Our architecture uses standard Web service tools and protocols; interface definitions specified using WSDL and communication using SOAP over HTTP. Site level managers oversee the overall performance of the components and ensure SLA compliance.

We see that progress must be made in several areas before an autonomic Web services architecture, such as the one described in this paper, can be deployed. First, Web service components are currently not, for the most part, autonomic. In fact, in

many cases, components require a complete shut-down and restart before configuration changes take effect, thus causing an interruption of service. Dynamic reconfiguration support is necessary for components to fit into an autonomic environment. As part of our research we are modifying open source Web based components, such as the Apache HTTP server, to enable dynamic configuration. Second, autonomic systems will require extensive monitoring, analysis and diagnosis. Most Web components currently provide sophisticated support to accomplish these tasks, however, ensuring that these processes do not burden the system with excessive overhead costs will be a challenge. Third, an architecture like the one proposed here relies on the specification of SLAs, goals and policies to determine acceptable performance. Users require a specification language in which these high level SLAs and policies can be expressed and SLAs must be translated into observable measures to be used as goals for each component. We plan to use the WSLA language [5] as the starting point and investigate how goals for individual components can be specified and derived from Web service SLAs.

## References


1. Apache Server Monitor, http://demo.freshwater.com/SiteScope/docs/ApacheServerMon.htm.
2. Birman, K., van Renesse, R., and Vogels, W.: Adding High Availability and Autonomic Behavior to Web Services, *26th International Conference on Software Engineering (ICSE'04)*, May 2004, Edinburgh, Scotland, United Kingdom.
3. Chiu, K., Web Services Performance: A Survey of Issues and Solutions, *7th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2003)*, Orlando, USA, July, (2003).
4. Fletcher, P., Waterhouse, M. (Eds).: Web Services Business Strategies and Architectures, Expert Press, (2002).
5. Dan, A., Davis, D., Kearney, R., Keller, A., King, R ., Kuebler, D., Ludwig, H., Polan, M., Spreitzer, M. and Youssef, A.: Web Services on Demand: WSLA-driven automated management. *IBM Systems Journal*, 43(1), (2004) 136 – 158.
6. J. A. Farrell, H. Kreger, Web Services Management Approaches. *IBM Systems Journal*, 41(2), (2002).
7. Ganek, A.G., Corbi, T.A.: The Dawning of the Autonomic Computing Era, *IBM System Journal*, V(42), N(1), (2003).
8. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. *Computer*, 36(1), (2003), 41-50.
9. Levy, R., Nagarajarao, J., Pacifici, G., Spreitzer, M., Tantawi, A.N., Youssef, A.: Performance Management for Cluster Based Web Services, *IFIP/IEEE 8th International Symposium on Integrated Network Management (IM 2003)*, (2003), 247-261.
10. Loosley, C., Gimarc, R.L., Spellmann, A.C.: E-Commerce Response Time: a Reference Model, *Keynote Systems Inc*., (2000).
11. Maes, P., Computational Reflection, *The Knowledge Engineering Review*, pp. 1-19, (1988).
12. Martin, P., Powley, W., Benoit, D.. Using Reflection to Introduce Self-Tuning Technology into DBMSs. *Proceedings of IDEAS'04*, Coimbra, Portugal, July 2004.
13. SOAP Version 1.2 Part 1: Messaging Framework, June 2004, http://www.w3.org/TR/soap12-part1/.

14. The Impact of Web Performance on E-Retail Success, Akamai Technologies, Feb. 1, (2004), http://www.akamai.com/en/resources/pdf/whitepapers/Akamai_eRetail_Success_Whitepaper.pdf.
15. Tian, M., Voigt, T., Naumowicz, T., Ritter, H., and Schiller, J.: Performance Impact of Web Services on Internet Servers, *International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, Marina Del Rey, USA, (Nov. 2003).
16. UDDI Version 3.0.1, UDDI Spec Technical Committee Specification, (Oct. 2003), http://uddi.org/pubs/uddi_v3.htm.
17. Web Services Description Language (WSDL) 1.1, (Mar. 2001), http://www.w3.org/TR/wsdl.
18. Weikum, G.: Self-tuning E-services: from Wishful Thinking to Viable Engineering, *High Performance Transaction Systems Workshop Submissions*, (Oct. 2001).
19. Wong, S.: Web services: The Next Evolution of Application Integration, http://www.eaiindustry.org/docs/WebServicesTheNextEvolutionofApplicationIntegration.pdf.
20. WebSphere Application Server Monitor, http://demo.freshwater.com/SiteScope/docs/WebSphereMon.htm.

## Appendix: WSDL Sample

The following shows the WSDL generated for the *setGoal* routine which is part of the Performance management interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://DefaultNamespace"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://DefaultNamespace"
xmlns:intf="http://DefaultNamespace"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <wsdl:message name="setGoalResponse">
      <wsdl:part name="setGoalReturn"
type="xsd:boolean"/>
   </wsdl:message>
   <wsdl:message name="setGoalRequest">
      <wsdl:part name="in0" type="xsd:string"/>
      <wsdl:part name="in1" type="xsd:double"/>
   </wsdl:message>
   <wsdl:portType name="Config">
      <wsdl:operation name="setGoal" parameterOrder="in0
in1">
         <wsdl:input message="impl:setGoalRequest"
name="setGoalRequest"/>
```

```
         <wsdl:output message="impl:setGoalResponse"
name="setGoalResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="ConfigSoapBinding"
type="impl:Config">
      <wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="setGoal">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="setGoalRequest">
            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="setGoalResponse">
            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="ConfigService">
      <wsdl:port binding="impl:ConfigSoapBinding"
name="Config">
         <wsdlsoap:address
location="http://webs2/axis/services/Config"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

# Extending UDDI with Recommendations: An Association Analysis Approach

Andrea Powles and Shonali Krishnaswamy

School of Computer Science and Software Engineering
900 Dandenong Road, Monash University, Caulfield East, Victoria –3145, Australia.
akapowles@gmail.com,
Shonali.Krishnaswamy@infotech.monash.edu.au

**Abstract.** This paper presents a novel recommendation extension to UDDI that we term RUDDIS. Recommendations can have potential benefits to both providers and consumers of Web Services. We adopt a unique technique to making recommendations that applies association analysis rather than traditional collaborative filtering approach. We present the implementation and demonstrate the functioning of RUDDIS in an unobtrusive manner where the user has total control over the recommendation process.

## 1 Introduction

Recommendations are used in a wide variety of e-commerce applications such as Amazon.com. Recommendations are useful for both buyers and sellers. For sellers, they provide a means to highlight additional products and for buyers they provide a filtered list of options to consider.

A natural and intuitive extension to the use of recommender systems in e-commerce is the investigation of such recommendations within web services. With the increasing recognition of the commercial potential of the service oriented paradigm, it is conceivable that recommendations within services can be of significant benefit and use. Providing recommendations for Web Services has the potential to offer many benefits for accessibility and usability of Web Services for both users and providers. It could present users with alternative or additional Web Service selections thus improving the likelihood that a web service will be consumed and that useful Web Services are being provided to users. Web Service providers could see an increase in the use of the Web Services they are providing as they would see greater exposure to possible users.

This paper presents a first investigation into incorporating a recommendation component within the web services framework. We propose a plug-in component to UDDI that extends the functionality of the UDDI from a discovery mechanism to one that performs discovery and recommendations for web service search queries. The focus on UDDI is evidently due its role as the standardised directory service component within current web services framework. As the UDDI specification has been designed with extensibility as a priority it is limited to only a few set functions. There have been numerous models and implementations for extending the UDDI

specification to address existing limitations including Rashid (2003), Lyell (2003), Systinet (2004), and Pokraev (2003).

While there have been many extensions of UDDI, there is yet to be any investigation into the use of recommendations for Web Services. With the expected increase of Web Service use, it will be beneficial for the UDDI to be able to provide recommendations of Web Services. Web Services would gain additional exposure through a UDDI that provides recommendations and users or systems looking to integrate or consume a particular Web Service would benefit as they are provided with additional Web Services that could be of use to them.

In developing a recommendation framework for service oriented architectures – we had two possible options: Automated Collaborative Filtering (ACF) (Herlocker 2000) and content-based approaches (Sarwar 2004). Automated Collaborative Filtering (ACF) is a widely used process for recommending information, services or physical items that are of potential use for a person based on ratings provided by other "similar" users. In content-based approaches the focus is on usage patterns rather than having a user focus. In ACF, similarity of users is typically based on maintaining user profiles. ACF is used in a wide variety of applications such as e-commerce (typically agent-based systems such as (Guttman 1998), recommendations for books [Amazon.com], music [CDNow.com] and movies [MovieFinder.com]. The key distinguishing feature of ACF as opposed to "content-based" approaches to making recommendations is the incorporation of the user dimension. However, there are several major challenges in developing ACF systems [HKR00]: the difficulty in developing valid user profiles, the question of mapping user profiles to individual preferences and tastes for varied items and services, the application of user ratings that do not capture the rationale for the ratings provided, the dependence on user ratings that tend to be subjective, and the requirement for users to provide additional information and perform extra tasks in providing the ratings.

The entire premise of ACF rests on the notion that similarity between users can be captured and represented. This premise is certainly valid when the objects in question are movies, books or music - but becomes intractable when the question pertains to web services. Consider questions such as: what makes two users invoke a particular weather information service as opposed to another? Building such user models for services is a worthwhile consideration – however, it requires considerable user psychology and usage patterns to be available in order to develop such user models. Furthermore, the dependence on users providing ratings is very obtrusive and the uptake of this - given the very limited incentives in a service oriented environment - is also questionable.

Therefore, this project aims to address these issues of ACF by using a content-based approach. This project proposes the use of association analysis [WiE99] to support recommendations in the web service environment. Association analysis or association rule mining is widely used as a data mining technique in the retail industry to perform tasks such as Market-Basket Analysis to search for interesting customer habits by looking at associations (Witten 1999). The classical example is the one where a store was reported to have discovered that people buying nappies tend also to buy beer. It is also used in applications in marketing, store layout, customer segmentation, medicine and finance. However, the value of using the concept of association analysis in a wider context is slowly emerging with applications in content-based image retrieval [MSP04]. The primary aim of association analysis is to

discover groups of items that occur together. Given a set of transactions {T}, each containing a subset of items from an item set $\{i_1, i_2, \ldots, i_m\}$, the focus is on the discovery of association relationships or correlations among a set of items. The strength of such associations is expressed by means measures known as *support* (i.e. the probability of a set of items occurring together ($P(i_j \cup i_k)$) and *confidence* (i.e the conditional probability of a set of items ($i_k$) appearing given that a set of items ($i_j$) exists ($P(i_k \mid i_j)$) ). The *support* indicates the frequency, while *confidence* denotes the strength of the association. In the context of service oriented environment, this technique alleviates many of the disadvantages highlighted with ACF, while retaining the strength that it is also like ACF derived from a user-centric basis. In association analysis, the transaction is derived from user activity – that is the user determines how services are invoked in conjunction with each other. This is the fundamental basis for performing an association analysis. On the other hand, it does not presume to build or rely on user profiles and identification of similarity between users, which is inherently challenging in the context of service oriented environments at this stage of its evolution. However, it may be foreseen that in future when such widespread user models and interactions are available ACF maybe used to in conjunction with content-based approaches (Pennock 2001)enhance results obtained through techniques such as association analysis. Furthermore, the occurrence of objects / items in a transaction is an easily documented event and there is no additional overhead in getting users to rate the objects / items they use. This "preferential rating" may easily be established through implicit means such as frequency and duration of usage by the same user.

We also note that the use of data mining techniques for recommendations in e-commerce (Schafer 2001) has been validated. The paper is organized as follows. In section 2 we present the design considerations and architecture of our UDDI extension to perform recommendations – RUDDIS. Section 3 presents the implementation of RUDDIS. Section 4 demonstrates its functioning using both a local and external UDDI. Finally section 5 concludes this paper.

## 2 Recommendations in UDDI (RUDDIS)

We are proposing the use of UDDI as a Recommender system. We term this model as Recommender Universal Description, Discovery and Integration System (RUDDIS). RUDDIS will consist of a UDDI registry encompassing a Recommendation component. This section examines the considerations and issues for the RUDDIS model. An UDDI that includes a Recommendation component should contain the following features:

- The model should conform to the UDDI specification and have minimal or no impact on the existing Web Services stack. The specification integrity being maintained is vital to the entire infrastructure and purpose of Web Services. Should the integrity of the specification be violated then interoperable nature strived for by Web Services may be foregone.
- Minimal effort should be required from the user to utilise RUDDIS compared to utilising a standard UDDI.

- The model should support the provision of useful and meaningful recommendations.

The key concern to be deliberated with regards to the design for the UDDI framework that includes a recommendation component is how to keep the UDDI compliant with the specification. The UDDI API Specification document provided by OASIS (Bellwood 2002) describes the programming interface and expected behaviours of all instances of the UDDI registry. When enhancing UDDI it is crucial to keep the standards set by this specification document. The UDDI data structures in the specification provide a framework for the description of basic service information, and an extensible mechanism to specify detailed service access information using any standard description language. Web Services are based on open standards which is the key to its heterogeneity. Altering the standards could damage the ability of others being able to use the Web Service stack. In this context, it is essential for RUDDIS to keep the recommendation and the UDDI components separate to ensure the compliance of the existing standards. The recommendation component and the UDDI component will be able to plug into each other via the calls made to and from RUDDIS. This way the UDDI will not require any internal modification and will maintain its integrity. This will allow the UDDI to function as normal. Service providers still wishing to register Web Services in the UDDI can do so as per usual. Clients wanting to search for Web Services without being provided with recommendations can do so with the RUDDIS model. With this transparency being modelled the user may never know they are using an extended UDDI.

In order for RUDDIS to provide useful recommendations we investigate the use of Recommendations using Market Basket Analysis. Market Basket Analysis is mainly used for data mining in the retail industry for discovering association rules between items in the data (Witten 1999). For example if we have a video shop that has a database of every hire transaction ever made over the history of the store. Each transaction contains customer details, the videos hired, how many and the video type e.g. Comedy, Romance, Horror, Drama or Action. As we mine through this data we find that in the cases where there was more than one video hired, selecting type Drama, 50% of the time also implies a comedy video was also hired. Then rule can be described as "drama" ➔ "comedy". Knowing such information can be very useful. In this case the store manager could place the Drama and Comedy sections closer together or introduce a special promotion for the two types when hired together. Association analysis is a relatively simple yet effective analysis tool and should be able to be implemented into a Recommendation System algorithm with ease. The Apriori algorithm (Witten 1999) or its many variants and enhancements are widely used as an effective implementation tool to support association analysis. It is simple and is computationally efficient. We propose to use Apriori for facilitation recommendation in RUDDIS.

We now examine how to ensure the model supports the provision of useful accurate recommendations. For the recommendations to be accurate, data being used to generate the rules needs to be accurate and up to date. RUDDIS is concerned with firstly how to obtain the data that will be used to generate the association rules then secondly, often the rules will be refreshed. Refreshing of the rules will require extra processing by the system which may slow down the performance. There is the need to weigh the importance of the performance of the system against the provision of the most accurate recommendations. We establish that to obtain the data required to

generate the association rules the users interactions with RUDDIS will be recorded. The details of all the queries made by users will be saved into a RUDDIS usage database. When the rules require refreshing the data from the RUDDIS usage database will be run through the Apriori algorithm to generate the rules. Also established is that the user should be able to determine the frequency of updating the rules. This way the user has control over the performance of RUDDIS as updating requires extra processing power. We also believe that it is essential to design this recommendation component such that it can be situated at the client or UDDI server for maximum flexibility.

### 2.1 RUDDIS Architecture

A scenario that could take place with the use of RUDDIS, is that a possible user is interested in searching for Web Services to do with planning a family trip to the east coast. Using RUDDIS, the user is looking up Web Services on airline flights to get there. The RUDDIS usage database contains all the Web Service requests made to RUDDIS and to which session it belonged. The RUDDIS usage database is utilised when recommendation rules need to be found. When our user enters in the query "flights", it is recorded in the RUDDIS usage database. Any other Web Service requests made by the same user at the one time will also be recorded under the same session. Also occurring in RUDDIS, is the data from the RUDDIS usage database being run through the Apriori algorithm producing a set of association rules. RUDDIS then seeks out any rules supporting "flights", if there are rules for this query item existing, the supporting rule with the strongest confidence level is found and the association item in that rule is extracted. So there maybe two rules for our query found such as "flights" ➔ "car hire" and "flights" ➔ "accommodation". Which ever rule of the two has the strongest confidence level for example the "flights" ➔ "accommodation" rule, gets the associated item extracted, in this example, "accommodation". The original query "flights" is then queried in the UDDI registry which contains the details of all registered Web Services. The association item extracted, "accommodation" is then also queried in the UDDI registry. For either of the two queries any Web Services found, are compiled and presented to the user, who may then decided to proceed integrating the Web Services.

The following five elements illustrated in Figure 1 have been identified as being required to carry out the tasks needed to be accomplished by RUDDIS.

**The Manager Component:** Is in control of handling all the interactions with the interface. As a Web Service request comes through the Manager Component evaluates the environment options selected by the user and the query item. It then directs the requests being made to the appropriate components. Any items returning from the other components are managed and acted upon by the Manager Component.

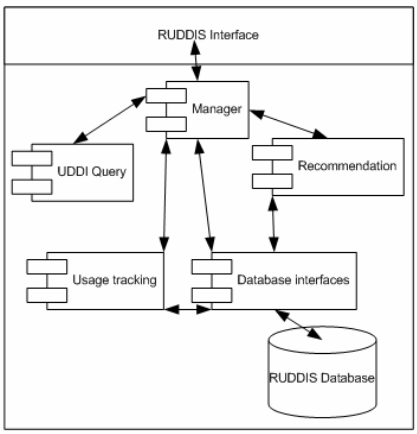



**Fig. 1.** RUDDIS Architecture

**UDDI Query Component:** Is in control of interacting with the UDDI registry which stores all the Web Services details. When passed a query item by the Manager Component it encapsulates the query into the appropriate format used for inquiries to the UDDI. It is supported by the UDDI API client framework which assists in the discovery of Web Services when requests are made. Any Web Services retrieved are then passed through to the Manager Component to deliver back to the user.
**Database interfaces Component:** Is in control of monitoring any databases within RUDDIS. Primarily this will be the RUDDIS usage database, but enables the provision of additional databases to be added to the system if this flexibility is required. Interactions between the Manager, Recommendation, Usage tracking and Databases interfaces Components occur when the tracking data is being recorded and when the RUDDIS usage database data is needed to run through the Apriori algorithm.

**Recommendation Component:** Is in control of discovering the association rules in RUDDIS and finding the strongest rule for the Web Service query being made by the user. If any results are found they are then passed back to the Manager Component to forward onto the UDDI Query Component, who sees if any correlating Web Services exist in the registry. It ensures the processes of extracting the data from the RUDDIS usage database and running the Apriori algorithm to generate the association rules.

**Usage tracking Component:** Is in control of ensuing that the users session and all the Web Services requested during the session are recorded. This data will be stored in the RUDDIS usage database through the use of the Database interfaces Component.

Each of the components have there own task which they are responsible for, but are required to communicate with each other to accomplish providing the recommendations to the user.

## 3 RUDDIS Implementation

A preliminary investigation of available UDDI implementations was required to select one for implementation. In order to determine which UDDI registry to utilise the following criteria was used in assessments. Based on the investigation we selected the use of the open source UDDI juddi supported by the use of UDDI4J (UDDI for Java) as the client. This selection was also based on recommendations being made for these two technologies being used together (UDDI.org, Hess 2004, Jung 2003).

The recommendation component in RUDDIS requires the ability to process the data from the Usage database using the Apriori algorithm. WEKA stands for the Waikato Environment for Knowledge Analysis. It provides practical machine learning tools and techniques with Java implementations (Witten 1999). WEKA contains an Apriori implementation which can be used to run the usage data through to find association rules.

The implementation was built using Java. The RUDDIS implementation was built as a web application using a combination of Java Server Pages (JSP) and Java Servlets running on the Jakarta Tomcat Server. For the describing of Web Services, they are categorised into two types, businesses and services. In RUDDIS the assumption is made that what we do for business can also be applied to services. For the implementation we have only the inquiry of businesses in the UDDI registry.

The Graphical User Interface (GUI) was implemented for RUDDIS takes the form of a web application to be used in a web browser that then accesses the juddi registry that resides on a server. The interface of RUDDIS was tailored to look like a standard UDDI interface with just some minor enhancements to assist with the recommendation section of the application and the facilitation of the selecting of various environment options. The interface is aimed to comply with the look and feel of existing public registries. Figure 2 provides a screen shot of the RUDDIS GUI.

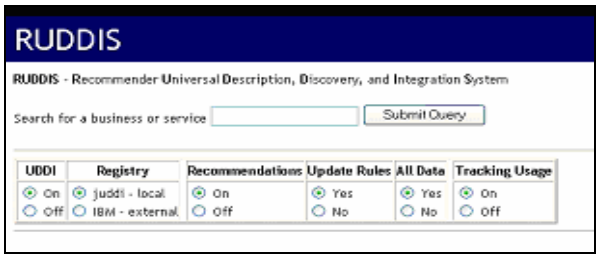



**Fig. 2.** RUDDIS GUI

## 4 RUDDIS At Work

The primary function of RUDDIS is to extend the UDDI to comprise of the ability to make recommendations of Web Services. Aside from this RUDDIS also allows the user to make various selections with regards to the recommendations being made. The section demonstrates the functioning of RUDDIS. This section illustrates the

feasibility of our approach and various options provided to the user to control the operation of RUDDIS such that the user has total control and the recommender system is as unobtrusive as possible. We now illustrate the options are as follows:

- A comparison of querying a local UDDI registry to querying an external registry for Web Services. The user is presented with the selection of Registry being either "juddi – internal" or "IBM – external". When juddi is selected the juddi UDDI registry on the local host is queried for Web Services. When IBM is selected the IBM test registry on an external server is queried for Web Services.
- A comparison of RUDDIS providing recommendations to RUDDIS working as a typical UDDI by not providing any recommendations. The user is presented with the selection of Recommendations being either "on" or "off". When "on" is selected, RUDDIS will attempt to provide any recommended Web Services found in the UDDI registry. When "off", RUDDIS will function as a typical UDDI providing only the Web Services found for the query item and not attempt to provide any recommendations.

For the purpose of assisting in the evaluation of RUDDIS, the juddi registry database was populated with an assortment of 315 web service names. The appropriate usage data was synthetically generated and used to populate the usage database to assist in the provision of recommendations. The usage database contains around 135 different sessions, each containing a number of Web Service requests. The RUDDIS usage database has been set up to assure that some rules supporting different scenarios will be generated. The evaluation data was used to generate the ARFF file required by WEKA. This lists the 24 rules discovered once the usage data is processed by WEKA. One of the rules generated supports the evaluation query of "skiing" ➔ "hire" meaning that a user looking for a Web Service on skiing would be highly likely to also want to look for Web Services on "hire". Other associations that could be useful that support this query are "lift passes", "snow reports" and "ski lessons". Rather than the user having to remember that these are items they could be interested in using, RUDDIS can offer them as recommendations.

The purpose of evaluating the difference between running with and without recommendations, is to compare RUDDIS in both scenarios. Not only do we want to see that associations analysis recommendations can be made with the methods that have been selected, but what the impact is on a typical UDDI in providing recommendations. When recommendations were turned on and "Skiing" was entered in as the Web Service query the following Web Service results produced included a recommended services list: *board hire, car hire, hire costs, ski hire, taxi hire, toboggan hire and hire snow gear.*

What is observed in the previous results is that RUDDIS searches through the rules and finds that for skiing, the strongest association rule contained the result of "hire". Under the Web Services Found heading, the Web Services retrieved from the registry that contain "skiing" are displayed. Under the Recommended Web Services heading any Web Services from the registry that contain "hire" are displayed. These were retrieved using find_business from UDDI specification. When recommendations are sected off the Web Service results are exactly the same as when recommendations are switched "on", except obviously no recommendations are provided. From examining these we can observe that the UDDI can be extended to provide Web Service recommendations. Also that it can be implemented in such a way that it can be requested ensure no recommendations are made.

We also evaluated to establish RUDDIS' ability to access an external UDDI registry. In this case the IBM test registry was used. Again the same query was entered. RUDDIS searched through the rules and found that for skiing, "hire" was the strongest rule result. It found no Web Services in the registry using the find_business library that contained "skiing". Under the Recommended Web Services heading any Web Services from the registry that contained hire are displayed. In this case there was one with the name of "Saphire". RUDDIS is dependant on what Web Services are registered in the external UDDI, so there were no Web Services existing in the IBM test registry that suited the query "skiing".

It can be determined from the above results that RUDDIS is successfully able to access an alternative external registry to the juddi UDDI on the local server and provide recommendations. The main difference is the Web Services retrieved as this is obviously a IBM test registry that contains a different set of registered Web Services.

## 5 Conclusions and Future Work

We have proposed and developed a recommendation extension to UDDI that we term RUDDIS. Recommendations can have potential benefits to both providers and consumers of Web Services. We have also adopted a novel approach to making recommendations that applies association analysis rather than traditional collaborative filtering approach. We have implemented and demonstrated the functioning of RUDDIS in an unobtrusive manner where the user has total control over the recommendation process. Further, we make no changes to the existing UDDI and the recommendation component acts as a plug-in that can used locally or at the server side.

We recognise that while we have highlighted the usefulness of this approach and demonstrated its practical feasibility – in order to fully validate such a model user trials that collect real data are essential. We recognise this as a limitation of our work so far. We plan to address these in at least a simulated context given that access to real usage data at this stage of web services research and development is not feasible. Furthermore, it is essential to determine the search space issues associated with a large list of recommendations. This notwithstanding, this paper takes the first step towards bringing the widely and successfully used concepts of recommendation in e-commerce to area of service oriented computing.

## References

1. Guttman, R, H., Moukas, A, G., and Maes, P., (1998), Agent-mediated electronic commerce: A survey, *Knowledge Eng. Rev.*, vol. 13, no. 2, pp. 147--159, 1998.
2. Herlocker, J, L. Konstan, J, A., and Reidl, J., (2000), Explaining Collaborative Filtering Recommendations, Proceedings of the 2000 ACM conference on Computer supported cooperative work**,** Philadelphia, Pennsylvania, USA**,** pp: 241 - 250
3. Hess Andreas, (19th March 2004), "How to set up your own UDDI registry" Andreas Hess [online] Available: http://moguntia.ucd.ie/programming/uddi/ [Accessed 2 June 2004]

4. Jung, Christoph. (15th October 2003), "Discovering and Publishing Web Services with JBoss.net" JBoss: Professional Open Source [online] Available: http://www.jboss.org/developers/guides/jboss.net/uddi [Accessed: 2nd June 2004].
5. Lyell M, Rosen L, Casagni-Sinkins M , Norris D, (2003), "On Software Agents and Web Services: Usage and Design Concepts and Issues", The MITRE Corporation [online] Available: http://www.agentus.com/WSABE2003/program/lyell.pdf [Accessed: 24th June 2004].
6. Padovitz, A., Krishnaswamy, S., and Loke, S, W., (2003), Towards Efficient and Smart Selection of Web Service Providers Before Activation , Proceedings of the Workshop on Web Services and Agent-based Engineering (WSABE 2003), [online] Available: http://www.agentus.com/WSABE2003/program/shonali.pdf [Accessed: 24th June 2004].
7. Pennock David, Lawrence Steve, Popescul Alexandrin, and Ungar Lyle, "Probabilistic Models for Unified Collaborative and Content-Based Recommendation in Sparse-Data Environments",In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-2001),* Morgan Kaufmann, San Francisco, 2001, pp. 437-444.
8. Pokraev, S. Koolwaaij, J. Wibbels, M. (2003, Aug, 4), "Extending UDDI with context-aware features based on semantic service descriptions", Xerox Corporation [online], Available: https://doc.telin.nl/dscgi/ds.py/Get/File-30562/ UDDI__paper__camera_ready.pdf [Accessed 4 August 2003].
9. Rashid,A.ShaikhAli,O.F.Rana,David.W.Walker (2003)"UDDIe: An Extended Registry for Web Services" [online], Department of Computer Science Cardiff University, UK Available: http://www.wesc.ac.uk/projects/uddie/uddie/papers/saint03.pdf [Accessed: 24th June 2004].
10. Sarwar B, Karypis G, Konstan J, Riedl J, (10th May 2001), "Item-based Collaborative Filtering Recommendation Algorithms", GroupLens Research Group/Army HPC Research Center [online] Available: http://www10.org/cdrom/papers/519/ [Accessed: 16th July 2004].
11. Schafer J Ben, Joseph A, Konstan, John Riedl, (2001 January - April), "E-Commerce Recommendation Applications", Data Mining and Knowledge Discovery, 5 (1-2): 115-153
12. Muller Henning, Squire David, Pun Thierry, (8th November 2003), "Learning from User Behaviour in Image Retrieval: Application of the Market Basket Analysis", *International Journal of Computer Vision*, vol. 56, no.(1/2/3), pp. 65-77.
13. "Systinet" (2004) New Features in Systinet UDDI registry 5.0, Systinet [online] Availbale: http://www.systinet.com/download/whats_new_in_wasp_uddi_5.0.pdf [Accessed: 29th July 2004]
14. "UDDI.org: UDDI Products and Components" (2nd April 2004) UDDI.org [online] Available: http://www.uddi.org/solutions.html [Accessed 2 June 2004].
15. Windley Phillip, (10th July 2003), Managing the Web services flow, InfoWorld [online] Available: http://www.computerworld.com.au/index.php?id=1765450771&fp=16&fpid=0 [Accesssed: 29th July 2004]
16. Witten Ian H, Frank Eibe, (October 1999), "Data Mining" *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations,* Morgan Kaufmann, p105-111.

# Ontology Based Model Transformation Infrastructure

Arda Goknil[1], N. Yasemin Topaloglu[2]

Department of Computer Engineering, Ege University, Izmir, Turkey,
[1]goknil@staff.ege.edu.tr
[2]yasemin@bornova.ege.edu.tr

**Abstract.** Using MDA in ontology development has been investigated in several works recently. The mappings and transformations between the UML constructs and the OWL elements to develop ontologies are the main concern of these research projects. We propose another approach in order to achieve the collaboration between MDA and ontology technologies. We propose an ontology based model transformation infrastructure to transform application models by using query statements, transformation rules and models defined as ontologies in OWL. Using this approach in model transformation infrastructure will enable us to use semantic web and ontology facilities in model driven architecture. This paper will discuss how these two technologies come together to provide automatization in model transformations.

## 1 Introduction

Model Driven Architecture (MDA) is a recent approach that has been introduced by OMG [10]. MDA considers model generation as the core activity of software development and specifically, it aims to accomplish software development through generating *Platform Independent Models (PIMs)* and mappings these models to *Platform Specific Models (PSMs)*. The main idea behind this is to enable software developers to work in a higher abstraction layer than the code level. As a consequence, models become the primary artifacts of software development [8]. To define mappings between models, *model transformation*, which takes one or more source models as input and produces one or more models as output, according to a set of transformation rules is needed.

An ontology is a formal explicit description of concepts in a domain of discourse, properties of each concept describing various features and attributes of the concept, and restrictions of slots [9]. Web Ontology Language (OWL) is a technology for ontology development and knowledge representation in Semantic Web [2]. OWL defines and instantiates Web ontologies. Recent works discuss that UML [12] could be a key technology for the ontology development bottleneck [1] [5] [6]. A number of partial solutions are currently available as a result of these works and Object Modeling Group (OMG) initialized a working group to create Ontology Definition Metamodel (ODM) to define M2 level UML-ontology-OWL transformation [13]. Alternatively to the established views, we propose another approach for the collaboration between MDA and OWL. While recent works discuss the contributions of MDA to

ontology development, we discuss the possible contributions of ontologies to MDA. We propose an ontology based model transformation infrastructure to transform application models by using query statements, transformation rules and models defined as ontologies in OWL.

In this paper, we discuss our ontology based model transformation approach and define the ontologies for model transformations within the context of MDA. We base our proposal on the idea that the current technologies for model transformations are not enough for interoperability of the model queries and transformation rules. The recent popular technologies to identify transformation rules are XMI and XSLT [15].

The paper is organized as follows. In Section 2, we discuss the general characteristics and underlying concepts of the ontology based model transformations. In Section 3, we introduce our approach and define the ontologies for model transformation infrastructure. Section 4 includes the conclusions.

## 2 Overview of Ontology Based Model Transformations

### 2.1 Web Ontology Language (OWL)

Web Ontology Language (OWL) is a technology to provide a standard language for the representation of ontologies on the web. OWL is a result of the ongoing process of defining a standard ontology web language. It is an extension of *Resource Description Framework* (RDF) [17]. OWL provides a rich set of vocabulary to catch all the relationships between classes and properties. An OWL document can include an optional ontology header and any number of class, property, and individual descriptions or axioms.

A Class identifier describes a named class in OWL ontology. For instance, "<owl:Class rdf:ID="Student">" defines a class "Student" which is an instance of "owl:Class". In the ontology, many individuals can be instantiated from the defined classes. Individuals are instances of classes, and properties may be used to relate one individual to another. These properties can be used to state relationships between individuals or from individuals to data values. For instance, an individual named *Olca* may be described as an instance of the class *Student* and the property *hasStudent* may be used to relate the individual *Olca* to the individual *EgeUniversity* which is derived from the class *University*. There are two kinds of properties defined in OWL: object property which relates individuals to individuals, and datatype property which relates individuals to data values. Similar to object-oriented programming, class hierarchies may be created by using one or more statements which shows that a class is a subclass of another class [2]. For instance, the class *UniversityStudent* is the subclass of the class *Student*. OWL allows restrictions to be placed on how properties can be used by instances of a class. This restriction mechanism in OWL provides to define constraints, which can not be specified in UML or other modeling techniques.

### 2.2 General Concepts of the Model Transformation Ontology

Model transformation is the core activity in MDA to generate new models or to change the existing models. A model transformation takes one or more source models as input and produces one or more models as output according to a set of transformation rules. The metamodeling technique is used to define these models and transformation rules [14]. A metamodel describes models by defining the meta entities and the relationships among these entities together with the semantics of these relationships. The meta class instances of the metamodel define the models and transformation rules generated from the metamodel. Extensible languages like *XML Metadata Interchange (XMI)* and *Extensible Stylesheet Language Transformations (XSLT)* can be used to encode models and transformation rules with meta class instances [3][15] [16].

XMI allows us encoding models in sets of XML tags to make them tool independent and interoperable. XSLT is another technology that enables to work on XML documents for model transformations. Though XMI and XSLT have reached wide usage, the interoperability and extendibility they provide are not sufficient. XMI is designed for interoperability among different case tools and it provides mechanisms for the exchange of UML models but it is not suitable for more structural interoperability.

The three main components of MOF 2.0 Query/Views/Transformations RFP [11] should be considered in the definition of model transformation ontology. The QVT RFP is issued by the Object Management Group (OMG) and seeks a standard solution for model manipulation. The three main subjects of model transformation defined by QVT [11]:

- Queries take a model as input, and selects specific elements from that model.
- Views are models that are derived from other models.
- Transformations take a model as input and update it or create a new model.

In our work, these three parts are defined as ontological. Defining queries and transformation in an ontology format will enable us to specify the structure of how meta entities and the relations between them are kept. Also queries defined in different transformation architectures will understand each other with the help of ontological approaches. To define instances from classes in XMI, you must define the meta classes in the same document. But in OWL, all instance queries reference a shared query ontology for the definitions of meta classes to define instances. The ontologies of these parts are defined as OWL documents. The definition in the OWL document provides a meta model for model transformations. For every instance transformation, instance ontologies can be derived from the meta ontologies.

## 3 Modeling the Transformation Components As Ontologies

As mentioned in QVT [11], the transformation infrastructure is constituted of three main structures as query, view and transformation. In our approach, we propose to model the meta entities and instances of these structures as ontologies.

### 3.1 Querying Application Models With Ontological Structures

Queries take a model as input, and select specific elements from that model. The aim is to detect the specific source and target patterns in application models. For that reason, different query languages have the same meta structures like selection and condition. These main structures are the basis of the query ontology.

Two different ontology documents are needed to query an application model. The first ontology document includes the meta classes of the query meta model. This meta model defines the main entities and the possible associations of these entities. The second document contains the instance query. The instance query selects the specific elements in the application document, and it is derived from the meta entities which are defined in the first query ontology. Figure-1 shows the relationship between the instance query ontology, meta query ontology, application model and the engine that process the query on the application model. *Query.rdf* includes the meta classes which constitute the meta model of model queries. These meta classes are the main selection elements like *Select, Where, And, Or, Not.* They associate the model elements to constitute the source and target patterns. *InstanceQuery.rdf* includes the instances of the classes in *Query.rdf* to define an executable query for an application model. *Query.rdf* is a kind of schema for query instances and defines the possible queries with its constructs.

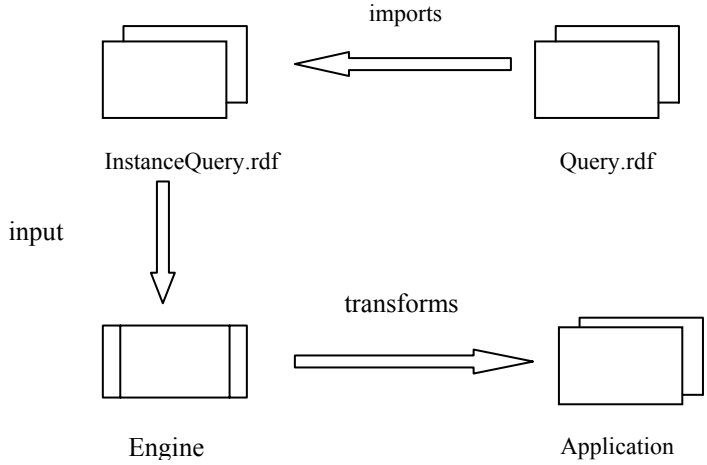



**Fig. 1.** Deriving Query Instances from Query.rdf

We propose a simple query language whose meta-model is shown in Figure-2[1]. The elements of this meta model constitute the structures in *Query.rdf*. The *Query* class in Figure-2 defines the query which is composed of two parts as *Where* and *Select*. The *Select* class associates with model elements which are derived after query processing. The *Where* class defines the condition in the query and is composed of the *Boolean terms (And, Or, Not), model elements* and *query references*.

[1] Instead of showing the OWL document, we model our ontology definition by using UML class diagrams because of the space limitation in this paper.

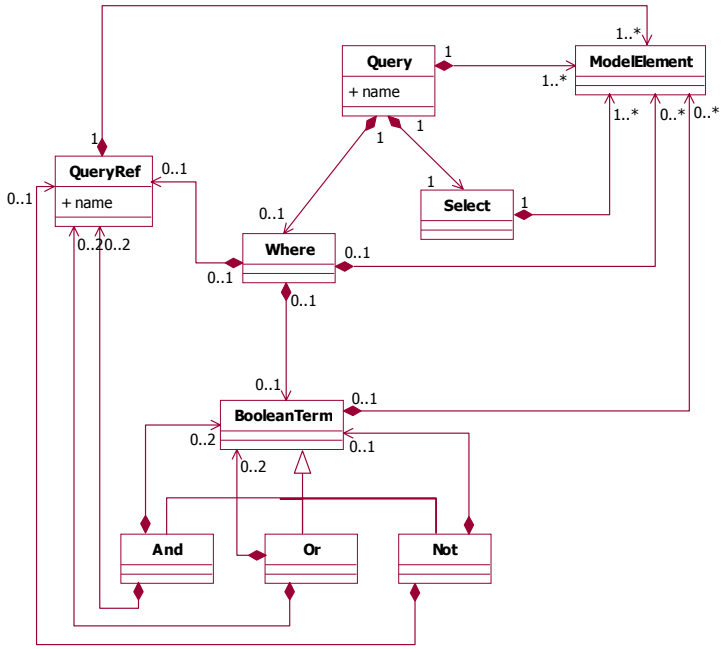



**Fig. 2.** The Meta Classes in Query.rdf

The *QueryRef* denotes other queries that are referenced in the *Where* term of the query. This enables us to use queries as recursive functions. The main difference between the *QueryRef* and the *Query* classes is that the *QueryRef* class is only a reference and does not contain the selection and the condition terms in the ontology where it is used. It defines the parameters which the *Query* it references uses in its own ontology. The Boolean terms include the model definitions as the conditions on the application model. The *Where* class may have these three classes in different combinations. The query ontology can limit the possible combinations that can be obtained from the meta model. The restrictions in the aggregation mechanism of the *Where* class and its collaborators can be defined in OWL as shown below:

```
<owl:Restriction>
       <owl:onProperty rdf:resource="#hasDefiniton">
         <owl:allValuesFrom>
              <owl:Class>
                   <owl:unionOf>
                     <owl:Class rdf:about="#QueryRef">
                     <owl:Class rdf:about="#BooleanTerm">
                     <owl:Class rdf:about="#ModelElement">
                   </owl:unionOf>
              </owl:Class>
        </owl:allValuesFrom>
      </owl:onProperty>
</owl:Restriction>

<owl:Restriction>
   <owl:onProperty rdf:resource="#hasDefinition"/>
  <owl:cardinality
```

```
  rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInt
eger">1</owl:cardinality>
  </owl:Restriction>
```

The property named *hasDefinition* defines the aggregation between the *Where* class and its collaborators, the *BooleanTerm* class, the *ModelElement* class and the *QueryRef* class. The first restriction defines that the *Where* class may have the *QueryRef*, the *BooleanTerm* and the *ModelElement* classes but with the second restriction it may have only one of them at once. The restriction mechanism in OWL provides to define constraints, which can not be specified in UML or other modeling techniques, for meta classes of our query model.

In Figure-2, we show the main classes that the query ontology must have. We can extend this ontology with additional structures for more complex model queries. For instance, there may be a set of same elements after the query processing. The query result set may have a model including a class associated with a set of same elements. To handle the set of model elements with iterations, there may be a container class to keep model elements as a set. A query which selects a class with a public attribute may return more than one public attribute of the class. We can handle the set of public attributes in the class in a set structure. Without a set structure, the query only matches the class with one public attribute at once. This set structure enables us to match one class with the set of its public attributes all at once.

Defining query models as ontologies allows us to extend this query meta model. The *InstanceQuery* ontologies are derived from *Query.rdf* for every model query like in Figure-1. In our work, *InstanceQuery.rdf* provides a query definition matching UML classes and their attributes, both owned by the class, and all of its superclasses. The query [4] shown below is an example of this.

```
QUERY hasAttr(C, A)
SELECT Class C, Attribute A, Class C2
WHERE A.owner=C OR (C.super=C2 AND hasAttr(C2, A))
```

It is possible for ontologies to be treated as reusable modules and imported into different documents. An OWL document may contain an individual of class defined in another ontology, which contains meta-data about that document itself. In our example, the *InstanceQuery* defining the *hasAttr* query imports *Query.rdf* to create individuals from the meta classes as shown below:

```
<owl:Ontology rdf:about="">
  <owl:imports rdf:resource="Query.rdf"/>
</owl:Ontology>
```

Every individual created in the *QueryInstance* references the class defined in the *Query* ontology. In our case, the *Where* individual has an *Or* individual. This *Or* individual has two properties named the *left-hand side* and the *right-hand side*. The left-hand side has a clause which defines *(A.owner=C)* and the right-hand side has an *And* individual. The *And* individual has *(C.super=C2)* clause in the left-hand side and a *QueryRef* referencing the *hasAttr* query with the parameters as *Class C2*, and *Attribute A*. Figure-3 shows this condition structure. In the ontology, we define *(A.owner=C)* clause with *Class C* which has *Attribute A*. Every model element used

in the query is defined and is aggregated by the *Query* individual. We use their references while defining the clauses in the conditions. It means that the reference of the *Class C* has the reference of the *Attribute A*. The reference mechanism allows us to define conditions on model elements by using temporary clauses. The model elements defined inside the query are accessed through their references while the conditions are defined.

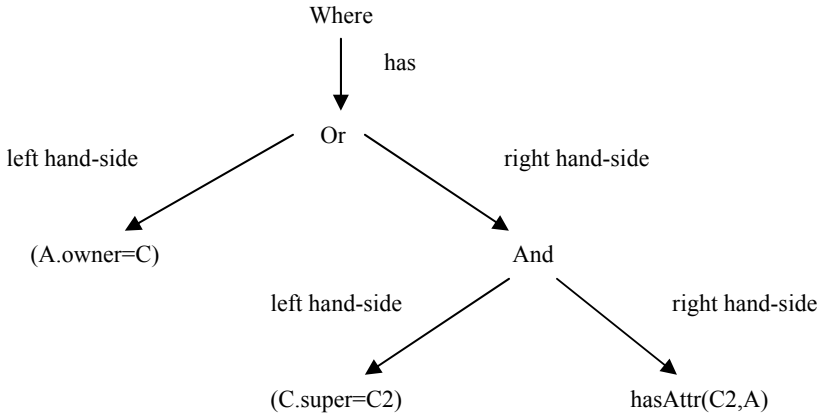



**Fig. 3.** The Structure of the Condition Statement in the Instance Ontology

### 3.2 Transforming Application Models With Ontological Structures

Transformations take a model as input and update it or create a new model. The submission for MOF 2.0 QVT RFP [11] split queries, views and transformations into two distinct groups. Queries and transformations may possibly create views, but views themselves are passive [11]. In our work, we consider that a transformation includes both queries and transformation operations. While queries select specific elements from the application model, transformation operations are applied to these selected model elements to transform the application model. The meta transformation ontology includes both the meta classes of transformation operations and queries. It can be considered that the transformation ontology is an extended query ontology to support the transformation operations.

The relationship between transformation ontology and transformations is similar to the relationship between meta ontology and instance ontology of queries that are discussed in Section 3.1. *Transformation.rdf* includes the meta classes which constitute the meta model of transformations and the instances in instance transformations are derived from these meta classes. *Transformation.rdf* is a kind of schema for transformation instances and defines the possible transformations with its constructs. Figure-4 shows the structures in our transformation ontology as a UML diagram.

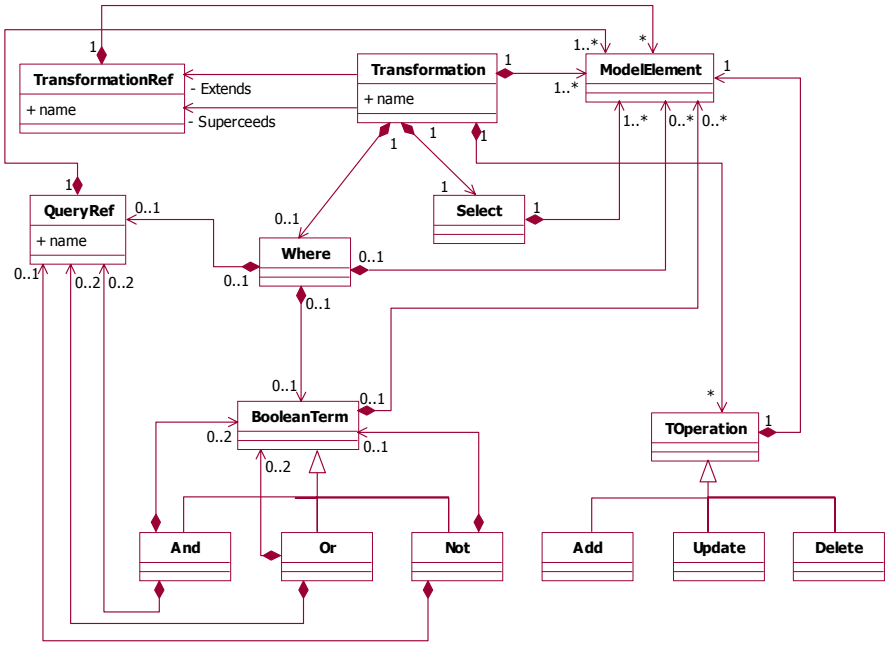



**Fig. 4.** The Meta Classes in Transformation.rdf

The *Transformation* class in Figure-4 defines the transformation itself. It has the *Select* and the *Where* classes like the *Query* class in *Query.rdf* because the transformation includes both model queries and transformation operations inside. The *Where* class is associated with the *QueryRef* class because a query instance can be referenced in the query condition of the transformation ontology. The query structures in *Transformation.rdf* and *Query.rdf* are the same. Transformations may be related to other transformations. [4] defines two ways for this relation: *Extends* and *Supersedes*. The associations between the *Transformation* and the *TransformationRef* classes in our ontology denote this relationship. Here, we have the OWL extensibility facilities to support other possible transformation relations in our ontology. Other possible relations between transformations can also be added to the transformation ontology in different approaches. The *TOperation* class and its sub-classes encapsulate the transformation operations on application models. The sub-classes of the *TOperation* class in our ontology are the *Add*, the *Update* and the *Delete* classes. We use them in our ontology to define the simplest operations. They are also the atomic operations and operate on meta class instances. In different and more complicated transformation approaches, more abstract and high-level classes may be used to define transformation operations.

Below, we give an example for a transformation definition which converts a public attribute from a given class to private and also creates the getter operation:

```
Transformation makingAttributePrivate(C, A)
SELECT Class C, Attribute A
WHERE (A.owner=C) AND (A.visibility="Private")
```

```
MAKE  A.visibility="Private"
      Define getMet = new Operation()
      getMet.name="getAttr"
      getMet.owner=C
```

The *Make* part after model query in the transformation defines the transformation operations on the application model. The first operation is making the visibility of *Attribute A* private. It is an update operation denoted by the individual derived from the *Update* class in the ontology. The second step in the transformation is creating a *get* method in *Class C*. An individual derived from the *Add* class does the creation of the get method named *getAttr* in *Class C*.

The third component of model transformations is views. Views are models that are derived from other models. Application models can also be defined in OWL instead of XMI where Case tools export and import the application models.

## 4 Conclusion

In this paper, we proposed an ontology based model transformation infrastructure. Ontologies provide a shared and common understanding of a domain. We consider the domain as model transformation in the context of model transformation languages and model the main constructs of model transformations. It enables us to extend our ontologies for future constructs of model transformations and allows communication of rules across applications. We used OWL in the definition of our ontologies because it is executable and is supported by tools. The restriction mechanism in OWL allows defining constraints about the instance models derived from meta models. OWL which provides to define assertions for UML has a precise semantics and is compared with *Object Constraint Language* (OCL) [18]. Some programmatic environments [7] include OWL APIs. These environments provide persistent storage, reading and writing OWL documents. Using OWL in model transformation infrastructure allows us to use the current semantic technologies to constitute the transformation engines. Loading and compiling the parts of model transformation are processed by the help of current ontology APIs.

Our aim is to investigate the possible contributions of ontologies to MDA and an ontology based model transformation infrastructure. We think that ontologies will play an important role in the development of MDA. In our future work, we will extend our transformation ontology with the constructs that support new transformation domains.

## References

1. Backlawski, K. et al: Extending the Unified Modeling Language for Ontology Development. Int. J. Software and Systems Modeling, Vol.1, No.2 (2002) 142-156
2. Dean, M., Schreiber, G. (eds): OWL Web Ontology Language Reference W3C Recommendation, Feb 10, 2004, http://www.w3.org/TR/owl-ref/

3. Demuth, B., Obermaier, S.: Experiements with XMI Based Transformations of Software Models. WITUML'01, Genova Italy, April (2001)
4. Duddy, K., Gerber, A., Lawley, M., Raymond, K., Steel, J.: Model Transformation: A declarative, reusable patterns approach. In Proceedings EDOC 2003, pp 174-185
5. Falkovych, K., Sabou, M., Stuckenschmidth, H.: UML for The Semantic Web: Transformation-Based Approaches. In Knowledge Transformation in Semantic Web, IOS Press, Vol.95 (2003), pp 92-106
6. Gasevic, D., Djuric, D., Devedzic, V., Damjanovic, V.: Approaching OWL to MDA Through Technological Spaces. WISME@UML'2004, Lisbon Portugal, 2004
7. Jena: A Semantic Web Framework For Java. http://jena.sourceforge.net/
8. Judson, S., France, R., Carver, D.: Specifying Model Transformations at the Meta-model Level. WISME@UML'2003, San Francisco USA, October (2003)
9. Noy, N., McGuinnes, D.: Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, March (2001).
10. OMG: MDA Guide Version 1.0.1. The Object Management Group, Document Number: omg/2003-06-01 (2003)
11. OMG: Submissions for MOF 2.0 Query/Views/Transformations Request for Proposal. The Object Management Group (2003)
12. OMG: OMG Unified Modeling Specification. Version 1.4. (2001)
13. OMG: Ontology Definition Meta-Model, http://www.omg.org/cgi-bin/doc?ad/03-08-06 (Current Apr 3, 2004)
14. Sendall, S., Kozaczynski, W.: Model Transformation – the Heart and Soul of Model-Driven Software Development. IEEE Software Sep/Oct. (2003), pp. 42-45
15. Staron, M., Kuzniarz, L.: Implementing UML model transformations for MDA. NWPER'2004, Turku Finland, August (2004)
16. Wagner, A.: A pragmatical approach to rule-based transformations within UML using XMI.difference. WITUML 2002, Malaga Spain, June (2002)
17. W3C Resource Description Framework. http://www.w3.org/RDF/
18. Zhao, Y., Assman, U., Sandahl, K.: OWL and OCL for Semantic Integration. Technical Report, Programming Environmental Lab (PELAB), Department of Computer and Information Science, Linköping University, Sweden.

# Evaluation of the Proposed QVTMerge Language for Model Transformations

Roy Grønmo[1], Mariano Belaunde[2], Jan Øyvind Aagedal[1], Klaus-D. Engel[3], Madeleine Faugere[4], Ida Solheim[1]

[1]SINTEF, Forskningsveien 1, Pb 124, Blindern N-0314 Oslo
{roy.gronmo, jan.aagedal, ida.solheim} @sintef.no
[2]France Telecom R&D, 2 Avenue Marzin, 22307 Lannion - France
mariano.belaunde@francetelecom.com
[3]Fraunhofer Gesellschaft FOKUS Kaiserin-Augusta-Allee 31, D-10589 Berlin
engel@fokus.fraunhofer.de
[4]THALES Research and Technology, Domaine de Corveville 91404 Orsay cedex - France
madeleine.faugere@thalesgroup.com

**Abstract.** This paper describes the set of requirements to a model-to-model transformation language as identified in the MODELWARE project. We show how these requirements divide into three main groups according to the way they can be measured, how to decompose them into different grades of support and how they can be weighted. The evaluation framework has been applied to the current QVTMerge submission which targets the OMG QVT standardization.

## 1 Introduction

Model-Driven Development (MDD) is a current buzzword that includes many technologies to improve the productivity in software development. Perhaps the greatest leap to make when adopting MDD is the shift from being code-centric to become model-centric. However, models will become first-class citizens only when there are suitable tools to ensure consistency and traceability between models on different levels of abstraction and from different viewpoints. A key concept in MDD is model-to-model transformation. Such model-to-model transformations define mappings between models, for instance to support refinement between models on different levels of abstraction. Model transformation makes it possible to derive models from other models in a controlled and automized manner. It also simplifies the way one relate models, for instance to ensure consistency. In the past few years many different proposals have been suggested for doing model transformations [1-3]. These heterogeneous solutions raise a need to standardize the way model transformations are performed. The OMG is currently finalizing a standard called QVT [4], for specifying model-to-model transformations, where the models are instances of metamodels defined using the Meta Object Facility (MOF) [5]. In this paper we evaluate the QVTMerge language [6], which is one of the two competing submissions towards the QVT standard.

This work has been conducted in context of MODELWARE, an EU-supported Integrated Project. An overall objective of MODELWARE is to improve productivity in software development. This objective will be pursued by realizing the vision of model-driven software development. To this end, model transformation is viewed as a crucial technology. MODELWARE includes both research institutions, tool vendors and end users, and this evaluation accommodates these different perspectives. We performed this evaluation to be able to produce model-to-model transformation technology that meets the requirements in MODELWARE, and we hope to influence the final stages of the ongoing standardization in OMG so that the standard meets the identified requirements.

We have identified a set of MODELWARE evaluation criteria for model-to-model transformation languages. Each criterion can be sorted in one of three categories according to how to test it:

- *Language inspection*. Manual inspection of the language alone is enough to evaluate the criterion.
- *Example-dependant*. In order to test such a criterion we need complete examples that show how the language is used in practice.
- *Tool-dependant*. Such a criterion requires a tool implementation.

Note that for some of the criteria it may be debated to which category they belong and if more than one category can be applied. Tools may implement additional functionality not provided by the language itself. However, less vendor and tool dependence is obtained if most of the criteria are satisfied by the language itself. Since no complete QVTMerge compliant tools are available we will not cover the tool-dependant criteria in this paper.

The criteria are presented in a template defining the rationale, scale, if the requirement is mandatory or optional, and weight. A rationale explains why the criterion is considered important, scale explains the different levels of support, and weight is a number between 1 (lowest importance) and 6 (highest importance). It is important that the scale is defined precisely and in a manner that it is easy to evaluate the target language. The importance level indicated by the weights is subjective and initial MODELWARE judgments. These weights are critical to ensure that evaluated languages are ranked higher if they fulfill the most important requirements.

## 2 Language Inspection Criteria

This section contains the list of criteria that can be tested by manual inspection of the inherent properties of the language. The criteria are sorted this way: mandatory requirements first, then higher weights first.

**Traceability** (mandatory, weight=5). *Rationale*: This property will make it easier for the user to understand how changes in the source will affect the target. It is also useful when undesired target results are produced, as the tracing back to the source element will be of important help in order to correct the source model or the definition of the transformation. *Scale*: 0 = No support, 1 = Manual support. The user must

explicitly express the elements to be traced. 2 = Automatic support. The language automatically provides traceability of all the elements.

**Unidirectionality** (mandatory, weight=4). *Rationale*: Unidirectionality is the ability to specify transformations in one direction only. When we never need to apply the reverse transformation it will be easier to concentrate only on the transformation one-way. *Scale*: 0 = no support, 1 = support.

**Complete textual notation** (mandatory, weight=4). *Rationale*: Textual notation enables users to define transformations without a graphical tool. Textual notations are also often preferred for defining large, complex transformations since graphical approaches are hard to scale. *Scale*: 0=no support, 1 = support.

**Black-box interoperability** (mandatory, weight=4). *Rationale*: This enables the reuse of any existing codes or scripts written in other languages, that otherwise would need to be rewritten in the transformation language. Support requires that it is possible to specify references to external code within a transformation. *Scale*: 0 = no support, 1 = support.

**Composition of transformations** (mandatory, weight=3). *Rationale*: This is desired in order to reuse several basic transformations to accomplish a more complex task. *Scale*: 0 = No support. 1 = Sequence only. 2 = Supporting the five basic control flow patterns [7] (sequence, and-split, and-join, or-split, or-join).

**Graphical notation** (mandatory, weight=2). *Rationale*: Graphical notations provide a higher-level view on the transformation and can more easily be communicated than a pure lexical alternative. *Scale*: 0 = No support. 1 = Only parts of a transformation can be graphical. 2 = A single transformation can be fully defined graphically. 3 = Compositions of transformations (see separate property) as well as single transformations can be fully defined graphically.

**Updating source model(s)** (mandatory, weight=2). *Rationale*: In some cases it is desired to update/complete an existing model instead of producing a new model. *Scale*: 0 = no support, 1 = support.

**Incomplete transformations completed with pattern parameters** (mandatory, weight=2). *Rationale*: This is a powerful construction for reusing large parts of a transformation that otherwise would need to be copied into several transformations. *Scale*: 0 = no support, 1 = support.

**Modularity** (optional, weight=6). *Rationale*: This will ease the comprehension and development of transformations. *Scale*: 0=no support, 1 = support. Support for this includes the possibility to split a transformation into several files, structure the code in separate UML package, provide separate transformation rules or to group methods inside classes, thus achieving fine grain modularity.

**Reusability** (optional, weight=5). *Rationale*: It is desirable to define transformations that capture common transformation rules that can be reused by other more specialized or parameterized transformations. This will improve the ability to share common knowledge, the ability to faster make new transformations and the ability to maintain the transformations. *Scale*: 0 = No support. 1 point for each of these that are satisfied: a) can import transformation library b) can specialize transformations. Maximum score is 2.

**Restricting conditions/pre-conditions** (optional, weight=4). *Rationale*: This is useful to ensure that the source model(s) provided to the transformation follows the restrictions set by the transformation. It prevents the transformation from being used

incorrectly and provides the opportunity to give critical feedback to the transformation user. *Scale*: 0 = no support, 1 = support.

**Bidirectionality** (optional, weight=2). *Rationale*: When a transformation needs to be defined in both directions as a relation between two models, it will be easier for the user to define one bidirectional transformation than to define two separate transformations for this purpose. The maintenance of a single transformation definition will also be easier to maintain and it reduces the risk of errors. *Scale*: 0 = no support, 1 = support.

**Multiple source models** (optional, weight=2). *Rationale*: The input from more than one source model may be necessary in order to produce the target. *Scale*: 0 = no support, 1 = support.

**Object orientation** (optional, weight=2). *Rationale*: The principles of object orientation will improve the reuse, maintenance and comprehension of transformations. *Scale*: 0 = No support. 1 point for each of these four OO principles that are satisfied: a) inheritance b) encapsulation c) identity/ instantiation d) late binding/ polymorphism. Maximum score is 4.

**Learning Curve** (optional, weight=2). *Rationale*: This property is desired since it increases the chance of becoming widely adopted. The weight is low, since it should not stop the introduction of a new way of programming style that has major advantages but that is unfamiliar to most people. *Scale*: Measured as an answer to the question: Is the transformation language easy to learn? (0 = Strongly disagree. 1 = Disagree. 2 = Neither. 3 = Agree. 4 = Strongly agree)

**Multiple target models** (optional, weight=1). *Rationale*: It may be desirable to produce more than one target model. *Scale*: 0 = no support, 1 = support.

## 3 Evaluating Ease-of-use Criteria by Examples

Most of the identified evaluation criteria were sorted in the language-inspection category and the tool-dependant category. Only two of the criteria were identified as being example-dependant: ease-of-use for simple and complex transformations. These two criteria are of high importance, and they require some case studies on reference transformation examples in order to be answered properly. The examples have been defined by an evaluation team and one of the authors of QVTMerge has assured that the language has been used in a suitable manner to solve the problem at hand. There are two alternative ways of defining transformations with QVTMerge. The first alternative uses predicate relations that declare the invariants that hold between source and target models (QVTMerge/Relations). The second alternative is a constructive directional approach based on operations (QVTMerge/Mappings). The evaluation has focused on the second approach.

All of the transformation examples have been defined using the concrete textual notation of the mapping formalism. The examples are Enterprise Java Beans/UML to Enterprise Java Beans/Java, XSLT to XQuery, UML Spem Profile to UML Spem Metamodel, UML to Relational Database, Book to Publication, and EDOC to J2EE. These examples cover both simple and complex transformations, vertical and horizontal, structural and behavioral transformation examples.

**Ease-of-use** (mandatory, weight=6). *Rationale*: This property is highly desirable in order to increase productivity and adoptability of a transformation language. *Scale*: Measured as an answer to the question: Is the transformation language easy to use? 0 = Strongly disagree. 1 = Disagree. 2 = Neither. 3 = Agree. 4 = Strongly agree. Important sub-questions that are useful to answer the main question: Is the transformation language clear and understandable? Does it require a lot of mental effort to set up the transformation? Is it easy to use the language to define transformations? Is it cumbersome to use? Is it frustrating to use? Is it controllable? Is it flexible?

None of the examples are fully presented in this paper due to limited space. Below is an extract from the EDOC [8] to J2EE (Java 2 Platform Enterprise Edition) transformation example. EDOC defines how to model enterprise systems using UML, while J2EE is a possible execution environment for EDOC models. This is a complex platform-independent model (PIM) to platform-specific model (PSM) transformation example.

```
module Edoc_To_J2EE (in edocModel:EDOC): j2eeModel:J2EE;
main () {
  edocModel.objects->firstPass();
  edocModel.objects->secondPass();
}
mapping firstPass(in EDOC::ModelElement) : JavaElement
  disjuncts Package_to_Package, ProcessComponent_To_Java_Interface {}
mapping secondPass(in EDOC::ModelElement) : JavaElement
  disjuncts
    PackageContainement,
    FlowPort_To_Method,
    Protocol_FlowPort_To_Method,
    OperationPort_To_Method, … {}
mapping PackageContainement[in EDOC.PackageDef]():J2EE.JavaPackage {
  init {
    result := self.resolveone(J2EE.JavaPackage);
  }
  subPackages := self.ownedElement[EDOC::PackageDef]
     ->resolveone(J2EE.JavaPackage);
}
```

The transformation specification uses two passes. The first pass is used to create the main structure and the data types, while the second pass is used to fill the detailed contents of the target model. The disjunction declaration in the second pass chooses separate rules for each target element to be created depending on the type of the source element. The `PackageContainment` rule transforms from EDOC packages to J2EE packages. The pre-defined `result` keyword is used to assign the target result object. `subPackages` refers to an association in the target metamodel which defines that J2EE packages may contain other J2EE packages. The built-in `resolveone` method is used to retrieve all target objects of a given type that were produced by a source instance in pass one. The final statement in the example assigns subPackages to a set consisting of J2EE packages that has already been transformed from EDOC packages in pass one.

When reviewing the example transformations some negative findings were discovered that may be used to further improve the specification before it is finalized as an OMG adopted specification:

- It is confusing when to use arrow and when to use dot for referencing part attributes/associations, built-in functions, inherited OCL functions etc.
- There is a mixture of procedural style with object-oriented style when defining and invoking methods. Object method calls are object-oriented (`theXSLTRoot.P2P`), while the signature uses an input parameter to represent the object type on which we can invoke the method like in the code extract signature above. This makes it non-intuitive to understand the much used "self" keyword that refers to the context parameter.
- It is hard to discover calls to the mappings rules. When doing transformations it is crucial to easily see where calls are made recursively or to other mapping rules. These calls cannot easily be distinguished from other calls to built-in functions, attribute/association references or OCL functions. XSLT has a solution for this by letting all calls to other mapping rules happen with the apply-templates instructions.

In addition to the negative findings described above, some issues were controversial because there were different opinions in the review group if the issues are negative findings or not:

**Long and cryptic expressions**. Single expressions are sometimes very long and cryptic to understand which requires a lot of mental effort. (Example: `return := out Return { expressions := self.nodes[#Template][t|t.match = '/']->nodes->flatten()->NodeToExpression();)` This is a heritage of OCL style and syntax. QVTMerge introduces additional short-hands to avoid excessive verbosity in single expressions – like the '#MyType' expression mapped as a call to the 'oclIsKindOf(MyType)' pre-defined operation . It is not clear yet whether these additional short-hands help on ease-of-use of the language. It is also possible for a transformation writer to split a computation in various lines using intermediate variables.

**Two-pass**. Some of the transformations use a two-pass approach in order to ensure that some target instances are produced so that the `resolve()` methods will get the proper element in a different context. This is a consequence of the explicit execution strategy in QVTMerge/Mappings which might be perceived as an advantage or as a disadvantage depending on writer preferences. An interesting issue here is to know whether it is possible to handle automatically object resolutions - so that the language user does not need to worry about this – without loosing the advantages of the explicit execution strategy.

The review of all the code examples shows nice program code structure, inheritance, and modularity by separation into manageable mapping rules. We believe that reusability and maintenance will be positive side-effects when the transformation code is written as they were in the examples. The example-based ease-of-use evaluation of the QVTMerge language shows slightly higher scores for complex than for simple transformations and the combination of vertical and structural transformations gets a lower score than the other categories of transformations. We need more examples in order to show that these trends are valid in general. But the overall average ease-of-use is evaluated as approximately 2.5 on a scale from 0 to 4, where 4 is the goal. It should be stressed that the evaluation of ease-of-use are subjective judgments of the MODELWARE participants who performed the example-based testing.

## 4 Related Work

The QVT Request for Proposal (QVT RFP) [4] identified a list of mandatory and optional requirements for submissions. Some of its requirements are focused on fitting the new QVT specification into the set of existing OMG specifications so to reuse and align well with existing recommendations. Many of the requirements of QVT RFP coincide with MODELWARE. The QVT RFP has identified portability and a declarative transformation language as requirements which are not directly stated by MODELWARE. There are several MODELWARE requirements not mentioned in the QVT RFP: object-orientation, composition of transformations, multiple source models, multiple target models, repetitiveness, black-box interoperability and modularity. The purpose of the MODELWARE requirements is to measure the goodness and quality of the approach regardless of any compliance with existing OMG recommendations.

Gardner et. al [9] and Langlois et. al [10] have reviewed the initial 8 submissions to the QVT RFP and proposed recommendations for the final specification. Most of their requirements are well covered already in this paper. Sendall and Kazaczynski [11] proposes these desired properties: executability, efficiency, fully expressiveness and unambiguity, clear separation of source model selection rules from target producing rules, graphical constructs to complement a textual notation, composition of transformations, and "conditions under which the transformation is allowed to execute". They propose that declarative constructions should be used for implicit mechanisms that are intuitive, but warns that too many implicit and complicated constructs may be more difficult to understand than the more explicit and verbose counterpart.

The way to measure ease-of-use in this paper is inspired by Davis [12] who suggests a decomposition of ease-of-use into sub-parts such as effort to become skillful, mental effort, error prone etc. These sub-parts can be answered on a scale ranging from strongly disagree to strongly agree. Davis has gone a lot further with his framework than we have done in this work, by showing how to organize these sub-parts, rank them, and use a questionnaire to compute final scores based on feedback from several reviewers. Krogstie [13] has proposed a framework for measuring the quality of models and modeling languages. Especially for graphical model transformation languages this framework should be applicable.

## 5 Evaluation Summary of QVTMerge

This section presents the evaluation of QVTMerge. In the table below the M (M=measured-scale-level) column shows the level of support and the S (S=score) column shows the weighted score for the criterion. The values in parentheses show the maximum value. Note that the level of support is downscaled to a value between 0 and 1 (0= no support, 1 = full support) by dividing by maximum scale level, which ensures that the criteria are treated on equal scales before the weights are applied. A final score can be computed by adding all the values in the S column. This is relevant to compare QVTMerge with competing model transformation approaches.

| Criterion | How it is supported by QVTMerge | M | S |
|---|---|---|---|
| Ease-of-use in simple transformations | See section 5. | 2.2 (4) | 3.3 (6) |
| Ease-of-use in complex transformations | See section 5. | 3 (4) | 4.5 (6) |
| Traceability | Fully automatic traceability is achieved by the four resolve operations that can trace from any source object to any target object and vice versa. | 2 (2) | 5 (5) |
| Unidirectionality | The language in textual as well as graphical notation directly supports it. | 1 (1) | 4 (4) |
| Complete textual notation | Any transformation can be fully defined with the mappings part in textual notation. | 1 (1) | 4 (4) |
| Black-box interoperability | A query operation, a mapping rule and transformation module may be declared without a body definition. This means that the implementation will be provided externally - for instance using Java. | 1 (1) | 4 (4) |
| Composition of transformations | QVTMerge does not get maximum score of 2 due to the lack of possibility to specify parallel control flows. | 1 (2) | 1.5 (3) |
| Graphical notation | The maximum score of 3 is not achieved due to lack of graphically specifying compositions such as “parallel split” and “synchronization” which is not possible at all. It is assumed that single transformations can be defined fully graphically although the specification states that in some complex transformations OCL annotations are needed. | 2 (3) | 1.3 (2) |
| Updating source model(s) | The transformation signature allows input parameters which can be specified as `inout`. | 1 (1) | 2 (2) |
| Incomplete transformations completed with pattern parameters | QVTMerge/Mappings: A mapping may extend "abstract" incomplete mappings.<br>QVTMerge/Relations: An abstract or checkable relation can be extended into executable transformations. | 1 (1) | 2 (2) |
| Modularity | The transformation may be grouped into several separate transformation rules. | 1 (1) | 6 (6) |
| Reusability | One point is given for the import module construction that enables one to import other libraries, and one point is given for the ability to specialize transformations by the extension mechanisms `extends`, `merges` and `inherits`. | 2 (2) | 5 (5) |
| Restricting conditions/preconditions | This is supported by associating the source model with a `modelType` with `complianceKind` = “strict”. | 1 (1) | 4 (4) |
| Bidirectionality | The textual relations part or the graphical notation enables bidirectionality. | 1 (1) | 2 (2) |

| | | | |
|---|---|---|---|
| Multiple source models | The transformation signature allows any number of input parameters. | 1 (1) | 2 (2) |
| Object orientation | Inheritance is supported by the three extension mechanisms `extends`, `merges` and `inherits`. Polymorphism is supported for query and mapping operations (through the virtual call mechanism). No specific mechanism is defined for object identity or encapsulation. | 2 (4) | 1 (2) |
| Learning Curve | One disadvantage is that there are many ways of doing the same thing, using relations, mappings, graphical or textual. It is however possible for a transformation writer to stick to a unique paradigm to minimize the learning effort. Another disadvantage is that there are many implicit constructions for shorthand notations that are hard to understand when you are a newcomer to this language. Advantages are that the textual language shares many similarities of both syntax and constructions with well-known object oriented languages such as Java and c#, c++. Furthermore the graphical notation is quite intuitive to understand. | 2 (4) | 1 (2) |
| Multiple target models | The transformation signature allows any number of output parameters. | 1 (1) | 1 (1) |

## 6 Conclusion and future work

This paper has identified 18 weighted evaluation criteria representing desired properties of a model-to-model transformation language. The list of requirements is more extensive than all of the previously published efforts. We have also gone further than previous efforts by defining six reference examples to measure the ease-of-use requirement which is of uttermost importance but requires such case studies in order to be measured. The evaluation of the current QVTMerge language shows that the mandatory requirement of transactional transformations is unsupported (such support is planned in a subsequent QVTMerge submission according the specification). Although QVTMerge achieves maximum scores for many of the criteria, we have revealed that the ease-of-use and learning curve of the QVTMerge language can be further improved. The MODELWARE evaluation criteria presented here is applicable to any model-to-model transformation language and can thus be used to rank model-to-model languages.

The advantages of QVTMerge are the modularity, black-box integration and nice structure of the program code into manageable separate transformation constraints and rules. Also we should point out the flexibility and openness, allowing a writer to select the kind of paradigm that is best appropriate to its transformation problem. We have also identified some disadvantages. Because there are many ways to define a transformation, using either the relations or mappings, textual or graphical, the learning curve for a user that would like to use all the possibilities, will be high. Many different programming styles can be used and mixed including imperative, declarative, object-oriented and procedural. All these options require more effort to be skilled and it may cause messy code if used incautiously. We have also experienced difficulties interpreting some of the single statements that are very long and cryptic. Such expressions are commonly used and they require a lot of mental effort.

An available QVTMerge tool is necessary to evaluate tool-dependant requirements such as performance, debugging functionality and robustness. Tool-dependant requirements have also been specified within MODELWARE, but are not presented in this paper due to limited space.

## Acknowledgement

MODELWARE is a project co-funded by the European Commission under the "Information Society Technologies" Sixth Framework Programme (2002-2006). Information included in this document reflects only the author's views. The European Community is not liable for any use that may be made of the information contained therein.

## References

1. Bézivin, J., et al., *The ATL Transformation-based Model Management Framework*. 2003, Université de Nantes: Nantes
2. Patrascoiu, O. *YATL: Yet Another Transformation Language*. in *First European Workshop on Model Driven Architecture with Emphasis on Industrial Application*. 2004. University of Twente, Enschede, the Netherlands.
3. Braun, P. and F. Marschall, *Transforming Object Oriented Models with BOTL*. Electronic Notes on Theoretical Computer Science, 2002. **72**(No. 3).
4. OMG, *Object Management Group MOF 2.0 Query / Views / Transformations RFP*. 2002,www.omg.org.
5. OMG, *Meta Object Facility (MOF) Specification*. 1997, Object Management Group,www.omg.org.
6. QVT-Merge_Group, *Revised submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10)*. 2004,www.omg.org.
7. Aalst, W.M.P.v.d., et al., *Workflow Patterns*. Distributed and Parallel Databases, 2003. **14**(3): p. 5-51.
8. OMG, *UML Profile for enterprise distributed Object Computing (EDOC) version 1.0; OMG Adopted Specification ptc/02-02-05*. 2002,http://www.omg.org/technology/documents/formal/edoc.htm.
9. Gardner, T. and C. Griffin. *A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard*. in *MetaModelling for MDA Workshop*. 2003. York, England, UK.
10. Langlois, B. and N. Farcet. *THALES recommendations for the final OMG standard on Query / Views / Transformations*. in *2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture*. 2003. Anaheim, California, USA.
11]. Sendall, S. and W. Kozaczynski, *Model Transformation – the Heart and Soul of Model-Driven Software Development*. IEEE Software, Special Issue on Model Driven Software Development, 2003.
12. Davis, F.D., *Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology*. MIS Quarterly, 1989. **13**(3): p. pp. 318-339.
13. Krogstie, J., *Evaluating UML Using a Generic Quality Framework*. UML and the Unified Process, ed. L. Favre. 2003: IRM Press

# Architectural Framework for Web Services Authorization

Sarath Indrakanti, Vijay Varadharajan, Michael Hitchens

INSS Research Group, Department of Computing
Macquarie University, Sydney, NSW 2109, Australia
{sindraka, vijay, michaelh}@ics.mq.edu.au

**Abstract.** This paper proposes an authorization architecture for Web services. It describes the architectural framework, the administration and runtime aspects of our architecture and its components for secure authorization of Web services as well as the support for the management of authorization information. The paper also describes authorization algorithms required to authorize a Web service client. The architecture is currently being implemented within the .NET framework.

## 1 Introduction

In general, security for Web services is a broad and complex area covering a range of technologies. At present, there are several efforts underway that are striving to provide security services for Web services. A variety of existing technologies can contribute to this area such as TLS/SSL and IPSec. There are also related security functionalities such as XML Signature and XML Encryption and their natural extensions to integrate these security features into Web service technologies such as SOAP [1] and WSDL [2].

WS-Security specification [3] describes enhancements to SOAP messaging to provide message integrity, confidentiality and authentication. The WS-Trust [4] language uses the secure messaging mechanisms of WS-Security specification to define additional primitives and extensions for the issuance, exchange and validation of security tokens within different trust domains. While there is a large amount of work on general access control and more recently on distributed authorization [5], research in the area of authorization for Web services is still at an early stage. There is not yet a specification or a standard for Web services authorization. There are attempts by different research groups [6-9] to define authorization frameworks and policies for Web services. Currently most Web service based applications, having gone through the authentication process, make authorization decisions using application specific access control functions that results in the practice of frequently re-inventing the wheel. This motivated us to have a closer look at authorization requirements for Web services and propose an authorization architecture.

In the next section, we describe our Web Services Authorization Architecture (WSAA). Section 3 discusses the benefits of the proposed architecture. We compare our architecture to related work in section 4 and then give some concluding remarks in section 5.

## 2 Web Services Authorization Architecture (WSAA)

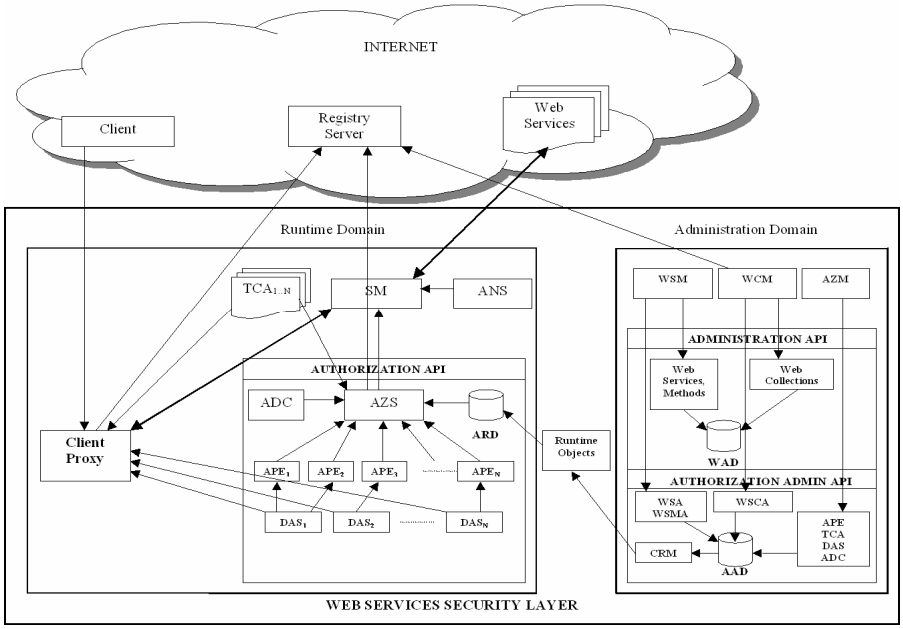



**Fig. 1.** Web Services Authorization Architecture (WSAA)

WSAA (figure 1) comprises of two domains - an administrative domain and a runtime domain. We manage Web services in the administration domain by arranging them into collections and the collections into a hierarchy. We provide administration support to manage a collection of Web services. We also provide support for the arrangement (adding, removing) of Web services within the collections and the movement of Web services within collections. Authorization related components such as authorization policy evaluators, trusted certification authorities (provide authentication and authorization credentials) and dynamic attribute services (provide attributes required for authorization) can be managed in the administration domain. Also security managers can assign a set of authorization policy evaluators to authorize requests to Web services.

To make the authorization process efficient, we have a runtime domain where the authorization related information such as what credentials are required to invoke a particular Web service and how to collect those credentials, is compiled and stored. This information is automatically compiled from time to time when necessary using the information from the administration domain and it can be readily used by components in the runtime domain.

The Registry Server located anywhere in the Internet is responsible for maintaining relations between services and their service providers. When a client requests the Registry Server for a specific service, the latter responds with a list of Web services that implement the requested service.

### 2.1 System Components

**Client Proxy (CP)** collects the required authentication and authorization credentials from the respective authorities on behalf of the client before sending a Web service request and handles the session on behalf of the client with a Web service's security manager.
**Security Manager (SM)** is an automated component responsible for both authentication and authorization of the client. A client's CP sends the necessary authentication and authorization credentials to the SM. SM is responsible for managing all the interactions with a client's CP.
**Authentication Server (ANS)** receives the authentication credentials from SM and uses some mechanism to authenticate the client. We treat ANS as a black box in our architecture as our focus in this paper is on authorization of the client. We included this component in the Web services security layer for completeness.
**Authorization Server (AZS)** decouples the authorization logic from application logic. It is responsible for locating all the authorization policy evaluators involved, sending the credentials to them and receiving the authorization decisions. Once all the decisions come back, it uses the responsible authorization decision composers to combine the authorization decisions. Where required, AZS also collects the credentials and attributes on behalf of clients from the respective trusted certification authorities and dynamic attribute services.
**Authorization Policy Evaluator (APE)** is responsible for making authorization decision on one or more abstract system operations. Every APE may use a different access control mechanism and a different policy language. However, an APE defines an interface for the set of input parameters it expects (such as subject identification, object information, the authorization credentials and dynamic attributes) and the output authorization result.
**Trusted Certification Authority (TCA)** is responsible to provide authentication and/or authorization credentials required to authenticate and/or authorize a client**.** For example, a TCA may provide public key certificates or authorization related certificates such as a Role Membership Certificate (RMC) [10].
**Dynamic Attribute Service (DAS)** provides system and/or network attributes such as bandwidth usage and time of the day. A dynamic attribute may also express properties of a subject that are not administered by security administrators. For example, a nurse may only access a patient's record if s/he is located within the hospital's boundary. A DAS may provide the nurse's 'location status' attribute at the time of access control. Dynamic attributes' values change more frequently than traditional static authorization credentials (also called privilege attributes). Unlike authorization credentials, dynamic attributes must be obtained at the time an access decision is required and their values may change within a session.
**Authorization Decision Composer (ADC)** combines the authorization decisions from authorization policy evaluators using an algorithm that resolves authorization decision conflicts and combines them into a final decision.

The *Authorization Manager* (AZM) for an organization is responsible to manage the APEs, TCAs, DASs and ADCs. S/he uses the Authorization Administration API for this purpose. The related data is stored in the Authorization Administration Database (AAD). See figure 1.

## 2.2 Web Services Model

We consider a Web service model based on the model defined in [7], where *Web Service*, *Web Service Method* and *Web Service Collection* are viewed as objects. Web service collections are used to group together a set of related Web service objects. Authorization related information can be managed in a convenient way if a set of related Web service objects is grouped together in a hierarchy of collections. Figure 2 shows an example of a hierarchy of Web service collections.

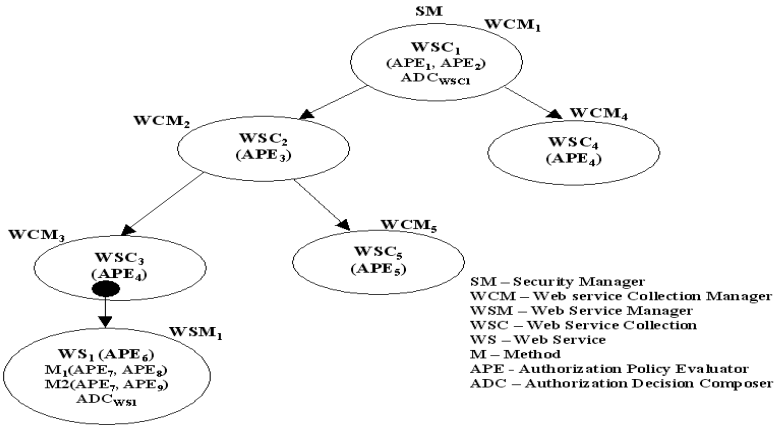



**Fig. 2.** Web Service Collection Hierarchy

## 2.3 Web Services Administration

A *Web Service Manager* (WSM) manages Web Services and Web Service Methods and a *Web service Collection Manager* (WCM) manages Web Service Collections using the Administration API (see figure 1). These objects are stored in the Web service Administration Database (WAD).

To effectively manage the collections, we arrange a set of related Web Service Collection (WSC) objects in a tree-shaped hierarchy as shown in figure 2. Each WSC in the hierarchy has a responsible Web service Collection Manager (WCM). There is only one Security Manager for a hierarchy of WSCs. In a WSC hierarchy tree, the root WSC's manager is called the *Root Web service Collection Manager* (RWCM). A RWCM is responsible for providing the Security Manager details (such as its location) in the WSDL statement of every Web service located under the collections s/he manages.

Let us consider an organization with a single hierarchy (such as the one shown in figure 2) of Web service collections. In figure 2, the root WSC is $WSC_1$ and the RWCM is $WCM_1$. We can consider a newly initiated system to simply consist of the root WSC, $WSC_1$ and a few Web Service (WS) objects under it managed by $WCM_1$. $WCM_1$ can add new WS objects from WAD into $WSC_1$. S/he can delete or move WS objects within the collections s/he is responsible for. There are other issues to consider such as 1) Who decides the location of a WS object (and how is the location changed)? 2) Who decides the shape of the tree itself? There are various design

choices to consider to answer these questions. Due to space limitations, we have not included the discussion on such design choices in this paper. We will describe these design aspects in a separate paper.

### 2.4 Authorization Data Administration and Policy Evaluation

A Web Service Manager (WSM) is also responsible to manage the authorization related information for the Web services s/he is responsible for. We consider a Web service method to be a high-level task that is exposed to clients. Each task (method) is made up of a number of system operations. These operations can be of different abstract types. For instance, each method of a Purchase Order service may perform one or more of these three operations - Web operation, Database operation and Mail operation. Each of these operations has a responsible authorization policy evaluator. It is reasonable to assume a WSM knows the set of tasks a Web service under his/her control performs. Similarly a WSM knows the set of operations each of these tasks (methods) perform. Using the authorization policy evaluator definitions from Authorization Administration Database (AAD), WSM associates authorization policy evaluators to Web services and their methods. This association is made in the *Web Service Authorization* (WSA) and the *Web Service Method Authorization* (WSMA) objects. WSM uses the Authorization Administration API to create and manage these objects. Similarly, a *Web service Collection Manager (WCM)* manages (using Authorization Administration API) authorization policy evaluator and authorization decision composer information in a separate object called *Web Services Collection Authorization* (WSCA) for all the collections s/he manages. These objects are stored in AAD.

Similar to Web service methods, a Web service can also have one or more authorization policy evaluators responsible for the Web service itself. Web service level policies are first evaluated before its method level authorization policies are evaluated. A Web service's authorization policy evaluators evaluate Web service level authorization policies. These policies will typically not be as fine-grained as method level authorization policies. A WSM may choose to create a new authorization decision composer for one or more Web services s/he manages or may decide to use one from the set of existing authorization decision composers from AAD if it serves the purpose.

Similar to Web services and their methods, a Web service collection can also have one or more authorization policy evaluators responsible for authorizing access to the collection itself. Collection level policies are first evaluated before a Web service's authorization policies are evaluated. A Web service collection's authorization policy evaluators evaluate collection level authorization policies. These policies will typically be course-grained when compared to Web service and Web service method level policies. Every root Web service collection has an authorization decision composer associated with it responsible for combining the decisions from all authorization policy evaluators involved. The coarse-grained authorization policies for all the relevant ancestor Web service collections (of an invoked Web service) are first evaluated, followed by the Web service level authorization policies and finally the fine-grained Web service method level policies are evaluated. The course-grained policies are first evaluated before the finer-grained policies as it helps reduce the

computing cost. If the client is not authorized by a course-grained policy, access can be denied straight away. For example in figure 2, when a client invokes $WS_1$'s method $M_1$, $WSC_1$'s authorization policies are first evaluated by APE1 and APE2, followed by $WSC_2$ ($APE_3$) and then $WSC_3$ ($APE_4$) policies. If $APE_1$, $APE_2$, $APE_3$ and $APE_4$ give out a positive decision, $WS_1$'s authorization policies are evaluated by $APE_6$. If $APE_6$ gives out a positive decision, then finally M1's authorization policies are evaluated by $APE_7$ and $APE_8$. $WS_1$'s authorization decision composer, $ADC_{WS1}$ combines the decisions from $APE_6$, $APE_7$ and $APE_8$ and if the final decision is positive, $WSC_1$'s authorization decision composer, $ADC_{WSC1}$ combines the decisions from $APE_1$, $APE_2$, $APE_3$, $APE_4$ and $ADC_{WS1}$. If the final decision from $ADC_{WSC1}$ is positive, the client will be authorized to invoke $WS_1$'s method $M_1$.

### 2.5 Runtime Authorization Data

We addressed who assigns (and how) authorization policy evaluators and authorization decision composers for Web services and Web service collections. The next question is, how does a client know, where necessary, how to obtain the required authorization credentials and dynamic runtime attributes before invoking a Web service? What are the responsible authorization policy evaluators (and the credentials and attributes they require), trusted certification authorities (the credentials they provide) and the dynamic attribute services (the attributes they provide)? How does the Authorization Server (AZS) know what the set of responsible authorization decision composers for a particular client request is?

To answer these questions, we have an Authorization Runtime Database (ARD) in the runtime domain. ARD consists of the runtime authorization related information required by clients and the Authorization Server. This information is exposed to clients in the form of authorization assertions defined in a *WS-Authorization Policy* statement attached to a Web service's WSDL statement. We define an XML schema for WS-Authorization Policy statement. The statement contains information about what credentials and attributes to collect and where to collect them from. However, we do not show the schema in this paper due to space limitation.

*Credential Manager (CRM)* is an automated component that creates and stores the authorization runtime information, in the form of objects in ARD, using the information from WAD and AAD databases. This makes the authorization process efficient as the information in ARD is streamlined for the runtime domain. CRM is invoked from time to time, when a Web service object is added or deleted to a collection, moved within a hierarchy of collections or when the shape of the tree itself changes, to update the runtime authorization information (objects) in ARD.

When a Web service object is placed and/or moved within a Web service collection in a tree, the set of authorization policy evaluators responsible for authorizing a client's requests changes. Similarly, the set of trusted certification authorities and dynamic attribute services responsible also changes. For example, in figure 2, when $WS_1$ moves from $WSC_3$ to $WSC_5$, the set of responsible authorization policy evaluators for $WS_1$'s method M2 changes from {$APE_1$, $APE_2$, $APE_3$, **$APE_4$**, $APE_6$, $APE_7$, $APE_9$} to {$APE_1$, $APE_2$, $APE_3$, **$APE_5$**, $APE_6$, $APE_7$, $APE_9$}. Once the change is made, CRM is automatically invoked and it updates ARD with the necessary runtime object entries for each method of $WS_1$. The responsible

authorization decision composers before and after the move will still be $ADC_{WSC1}$ and $ADC_{WS1}$.

### 2.6 Authorization Algorithms

WSAA supports three algorithms. The first, *push-model* algorithm supports authorizations where a client's Client Proxy, using WS-Authorization Policy, collects and sends the required credentials (from trusted certification authorities) and attributes (from dynamic attribute services) to a Web service's Security Manager. The second, *pull-model* algorithm supports authorizations where the Authorization Server itself collects the required credentials from trusted certification authorities and authorization policy evaluators collect the required attributes from dynamic attribute services. The third, *combination-model* supports both the push and pull models of collecting the required credentials and attributes.

An organization must deploy one of these algorithms depending on the access control mechanisms used. If all the access control mechanisms used by the set of authorization policy evaluators are based on a pull model, then the organization must deploy the pull-model algorithm. If all the access control mechanisms used are based on a push model, then the organization must deploy the push-model algorithm. However, when some of an organization's authorization policy evaluators use the pull-model and others use the push-model, the combination-model algorithm must be deployed.

## 3 Discussion - Benefits of the Proposed Architecture

Some of the key advantages of the proposed architecture are as follows:
**(a)** Support for various access control models: WSAA supports different access control models including mandatory access control, discretionary access control, role-based access control, and certificate based access control models. The access policy requirements for each model can be specified using its own policy language. The policies used for authorization can be fine-grained or coarse-grained depending on the requirements. Access control mechanisms can either use the push-model or pull-model or even a combination of both for collecting client credentials.
**(b)** Support for legacy applications and new Web service based applications: Existing legacy application systems can still function and use their current access control mechanisms when they are exposed as Web services to enable an interoperable heterogeneous environment. Once again different access policy languages can be used to specify the access control rules for different principals. They could adopt a push or a pull model for collecting credentials. At the same time WSAA supports new Web service based applications built to leverage the benefits offered by Web services. New access control mechanisms can be implemented and used by both legacy and new Web service applications. A new access control mechanism can itself be implemented as a Web service. All WSAA requires is an end-point URL and interface for the mechanism's authorization policy evaluator.
**(c)** Decentralized and distributed architecture: WSAA allows a Web service to have one or more responsible authorization policy evaluators involved (each with its own

end-point defined) in making the authorization decision. The authorization policy evaluators themselves can be defined as Web services specializing in authorization. These features allow WSAA to be decentralized and distributed. Distributed authorization architecture such as ours provides many advantages such as fault tolerance and better scalability and outweighs its disadvantages such as more complexity and communication overhead.

**(d)** Flexibility in management and administration: Using the hierarchy approach of administering Web services and collections of Web services, authorization policies can be specified at each level making it convenient for Web service collection managers (WCM) and Web service managers (WSM) to manage these objects as well as their authorization related information. Another benefit of WSAA is that the credential manager component automatically generates runtime authorization objects.

**(f)** Ease of integration into platforms: Each of the entities involved both in administration and runtime domains is fairly generic and can be implemented in any middleware including the .NET platform as well as Java based platforms. The administration and runtime domain related APIs can be implemented in any of the available middleware.

**(g)** Enhanced security: In our architecture, every client request passes through the Web service's security manager and then gets authenticated and authorized. The security manager can be placed in a firewall zone, which enhances security of collections of Web service objects placed behind an organization's firewall. This enables organizations to protect their Web service based applications from outside traffic. A firewall could be configured to accept and send only SOAP request messages with appropriate header and body to the responsible security manager to get authenticated and authorized.

## 4 Related Work

Kraft proposes a model based on a "distributed access control processor" for Web services [7]. The main components in the authorization model are the gatekeeper, which intercepts SOAP requests to a Web service and one or more Access Control Processors (ACPs) that make the authorization decisions for the Web service. The gatekeeper itself can be an ACP. It also has the responsibility of authenticating the requesting client, combining the decisions from individual ACPs and to make the final access control decision. The advantage of this model is it supports decentralized and distributed architecture for access control. The model is generic enough to support different models of access control. This model however, does not provide support for administration of authorization related information. It also does not provide support to manage Web service collections and their authorization related information using standard APIs, which our architecture provides.

Yague and Troya [8] present a semantic approach for access control for Web services. The authors define a Semantic Policy Language (SPL). SPL is used to create metadata for resources (Secure Resource Representation (SRR)) and generic policies without the target resource in them. A separate specification called Policy Applicability Specification (PAS) is used to associate policies to target objects at run time dynamically when a principal makes a request. The architecture is based on the

integration of a Privilege Management Infrastructure (PMI) and the SPL language features. At run time, depending on the Source of Authorization Descriptions (SOADs) that the Source of Authorization (SOA) in the PMI is willing to provide to the client and the SRRs, the Policy Assistant component streamlines the SPL policies and the PAS. What is interesting in this model is that the authorization policies can be attached dynamically based on the metadata of the resource being accessed and also be streamlined dynamically to the SOADs the SOA is willing to send, through the PMI client. The disadvantage with this model is that authorization policies can only be written in SPL and is based on one model of access control – the PMI, which means this model is not generic enough to support different access control mechanisms required by applications in a heterogeneous environment. This means unlike our architecture, legacy applications (using their own access control mechanisms) are not supported by this model. The model also does not provide management and administration support for Web service objects.

Agarwal et al [6] define an access control model that combines DAML-S [11], an ontology specification for describing Web services and SPKI/SDSI [12], used to specify access control policies and to produce name and authorization certificates for users. Access Control Lists (ACLs) are used to specify access control policies of Web Services. Each ACL has the properties keyholder, subject, authorization, delegation and validity. Access control is defined as a pre-condition to access a Web service. When trying to access a Web service, a user sends the set of credentials needed to access the Web service. The user does this by using the ACL provided in the access control precondition of the Web service provider. The user calculates the set of certificates needed by making use of a chain discovery algorithm. If the client is authorized with the certificates provided, the Web service returns the functional outputs sought by the client. This model is a certificate based access control model and so is not generic enough to support multiple access control models. This means legacy applications exposed using Web services cannot use different models of access control they have already been using. The ACLs in this model are simple and one cannot specify fine-grained and complex authorization policies using this model. The model also does not provide management and administration support for Web service objects.

Ziebermayr and Probst discuss their authorization framework [9] for “simple Web services”. Their framework does not consider distributed authorization and assumes that Web services provide access to data or sensitive information located on one server and not distributed over the Web. The framework uses a rule based access control model where simple rules are written for components (in which Web services reside), Web services and parameters of a Web service method. A rule consists of a reference to a service definition, another reference to a user and additional rule information for parameters where necessary. When an access request comes in, the rules at these various levels are checked and an authorization decision is made. This framework uses simple rule based access control and so does not support different models of access control. This means legacy applications cannot be exposed as Web services. Another disadvantage with this framework is that it cannot support authorizations for distributed Web services, which have access to data and/or information over a number of Web servers. Unlike our architecture, there is no abstraction of each Web service method’s function into a set of operations. This abstraction makes it easy to perform authorization administration as discussed earlier.

## 5 Concluding Remarks

We proposed an authorization architecture for Web services - WSAA. We described the architectural framework, the administration and runtime aspects of our architecture and its components for secure authorization of Web services as well as the support for the management of authorization information. WSAA supports push-model, pull-model and combination-model authorization algorithms.

The architecture supports legacy applications exposed as Web services as well as new Web service based applications built to leverage the benefits offered by Web Services; it supports old and new access control models and mechanisms; it is decentralized and distributed and provides flexible management and administration of Web service objects and authorization information. We believe that the proposed architecture is easy to integrate into existing platforms and provides enhanced security by protecting exposed Web services from outside traffic. We are currently implementing the proposed architecture within the .NET framework.

## References


1. World Wide Web Consortium (W3C), "SOAP v1.2, http://www.w3.org/TR/SOAP/," 2003.
2. World Wide Web Consortium (W3C), "Web Services Description Language (WSDL) v1.1, http://www.w3.org/TR/wsdl," 2001.
3. B. Atkinson et al, "Web Services Security (WS-Security) Specification, http://www-106.ibm.com/developerworks/webservices/library/ws-secure/," 2002.
4. S. Anderson et al., "Web Services Trust Language (WS-Trust), http://www-106.ibm.com/developerworks/library/specification/ws-trust/," 2005.
5. V. Varadharajan, "Distributed Authorization: Principles and Practice," in *Coding Theory and Cryptology, Lecture Notes Series, Institute for Mathematical Sciences, National University of Singapore*: Singapore University Press, 2002.
6. S. Agarwal, B. Sprick, and S. Wortmann, "Credential Based Access Control for Semantic Web Services," *American Association for Artificial Intelligence*, 2004.
7. R. Kraft, "Designing a Distributed Access Control Processor for Network Services on the Web," presented at ACM Workshop on XML Security, Fairfax, VA, USA, 2002.
8. M. I. Yagüe and J. M. Troya, "A Semantic Approach for Access Control in Web Services," presented at Euroweb 2002 Conference. The Web and the GRID: from e-science to e-business, Oxford, UK, 2002.
9. T. Ziebermayr and S. Probst, "Web Service Authorization Framework," presented at International Conference on Web Services (ICWS), San Diego, CA, USA, 2004.
10. J. Bacon and K. Moody, "Toward open, secure, widely distributed services," *Communications of the ACM*, vol. 45, pp. 59-64, 2002.
11. M. B. A. Ankolekar, J. R. Hobbs,O. Lassila, D. McDermott, D. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, "DAML-S: Web Service Description for the Semantic Web," presented at 1st International Semantic Web Conference (ISWC), Sardinia, Italy, 2002.
12. C. M. Ellison, B. Frantz, B. Lampson, R. L. Rivest, B. M. Thomson, and T. Ylonen, "Simple public key certificate, http://theworld.com/~cme/html/spki.html," 1999.

# Towards a formalization of model conformance in Model Driven Engineering

Thanh-Hà Pham[1,2], Mariano Belaunde[1], Jean Bézivin[2]

[1] France Télécom R&D, 2 avenue Pierre Marzin, 22300 Lannion Cedex, France
{thanhha.pham, belaunde.mariano}@rd.francetelecom.com
[2] ATLAS Group, INRIA&LINA, University of Nantes, France
jean.bezivin@lina.univ-nantes.fr

**Abstract.** The principle of "*everything is an object*" basically supported by two fundamental relationships *inheritance* and *instantiation* has helped much in driving the object technology in the direction of simplicity, generality and power of integration. Similarly in the Model Driven Engineering (MDE) today, the basic principle that "*everything is a model*" has many interesting properties. The two relations *representation* and *conformance* are suggested [2] to be the two basic relations in the MDE. This paper tends to support this ideas by investigating some concrete examples of the *conformance* relation concerning three technological spaces (TS) [10]: Abstract/Concrete Syntax TS, XML TS and Object-Oriented Modeling (OOM) TS. To go further in this direction we try to formalize this relation in the OOM TS by using the category theory – a very young and abstract but powerful branch of mathematics. The OCL language is (partially) reused in this scheme to provide a potentially useful environment supporting MDE in a very general way.

## 1 Introduction

Model Driven Engineering (MDE) today does not limit itself to the OOM Technological Space (TS) but many other TSs such as AS TS, XML TS ... [10]. This means explicitly that its principles must be very general and not only restricted to OOM TS. Today, the principle « Everything is a model » as suggested by many authors such as [3] becomes the main principle of the MDE similarly to the principle « Everything is an object » in object technology. Conformance is one of the fundamental relations supporting this principle in MDE. This paper investigates the conformance relation in some well-known Technological Spaces such as Abstract/Concrete Syntax, XML and OOM technological spaces.

The paper is organized as follow: section 1 presents the context of our work; section 2 presents some ideas about the notion of conformance in several well-known TSs; section 3 presents a formalization of the conformance relation in the OOM TS using category theory and the OCL language. The practical usage of this formalization will be discussed in the section 4. Some related works are briefly introduced in the section 5. Some conclusions will be provided in the section 6.

## 2 Conformance in some Technological Spaces

We begin our discussion with a simple example coming from Regular Expression. It is not difficult to see that there is a mapping from a string S = acccd to a regular expression E = a(b|c*)d? when the string S matches the expression E. This mapping is illustrated in the Fig.1.

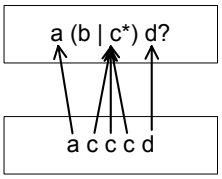



**Fig. 1.** A very simple form of conformance – a string matches a regular expression

The regular expression E defines characters that may appear in a string conforming to E: {a,b,c,d} and how these characters are structured using several constructions:

- *alternation* with a vertical bar such as b | c specify the choice of b or c.
- *quantification* with a quantifier (+,?,*) that following a character specifies how often that character is allowed to occur.
- *grouping* with brackets to define the scope and precedence of the other operators.

If the guiding principle of the MDE:

"**Everything is a model**" [P0]

is accepted, we have the following two models: the string S and its definition E (is also a string) with their characters as model elements. It can be said that S is defined by E or S conforms to E.

"*A model conforms to its definition, this definition is also a model called meta-model of the first one*" [P1]

From our first observation, we propose the following principle:

"*Every element of a model finds an unique definition in a meta-model that the model conforms to*" [P2]

We have also the following comments:

- The order of elements in S must respect to the order of elements defined in E. [C1]
- The group of elements in S must respect to the group definition in E. [C2]
- The number of occurrences of elements in S must respect to quantification definitions in E. [C3]

Now we move to an illustrative example in the Abstract/Concrete Syntax TS. Let's consider a well-known *HelloWorld* program written in the Pascal programming language. This program is considered to be a syntactically correct with respect to the grammar of the Pascal programming language. In this example, the *HelloWorld* program is a model and the grammar of the Pascal programming language is the meta-model defining the former. The principle [P2] is applicable in this case and is illustrated in the Fig.2. A part of the grammar is represented in the flowchart form extracted from [9]. Every symbol of this program finds a unique definition in the grammar. The three comments [C1, C2, C3] are also correct in this case.

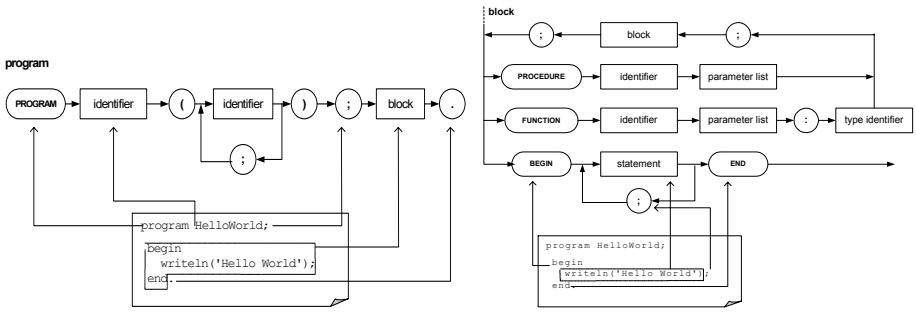



**Fig. 2.** A Pascal program conforms to the grammar of the Pascal programming language

In the XML TS, we find the following definition [6]: « *An XML document is valid if it has an associated document type declaration and if the document complies with the constraints expressed in it* ». This means explicitly that a valid XML document must conform to a DTD. DTDs specify two kinds of constraints as classified in [5]: *structural* constraints given by element declaration rules and *attribute* constraints given by attribute declaration rules. Also following [5], « *the structural constraints of DTD are abstracted as extended context free grammars, that is, context free grammars where the right hand side of each production contains a regular expression. An XML document is valid with respect to the structural constraints of a DTD if its abstraction as a tree represents a derivation tree of the extended CFG corresponding to that DTD* ». Attribute constraints deal with the values of attribute nodes while structural constraints deal with the labels of nodes in the XML tree.

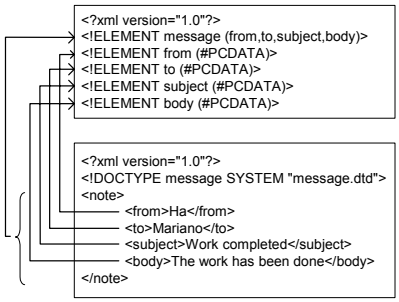



**Fig. 3.** An XML document conforms to a DTD.

Let's consider an example that illustrates the relation between an XML document and a DTD. In this case, the model is the XML document and the meta-model defining this model is the DTD. The XML document has (element and attribute) nodes as its elements. The principle [P2] and the three comments [C1, C2, C3] are also applicable in this case.

We have analyzed the conformance relation in the case of regular expression, Abstract/Concrete syntax and XML. The principle [P2] is also applicable in Object-Oriented modeling.

In the left of the Fig.4 is an UML diagram represented in a case tool such as Rose. This model is an *instance-of* of the UML meta-model as simplified in Fig.4. Every elements of this model finds its unique definition in the meta-model.

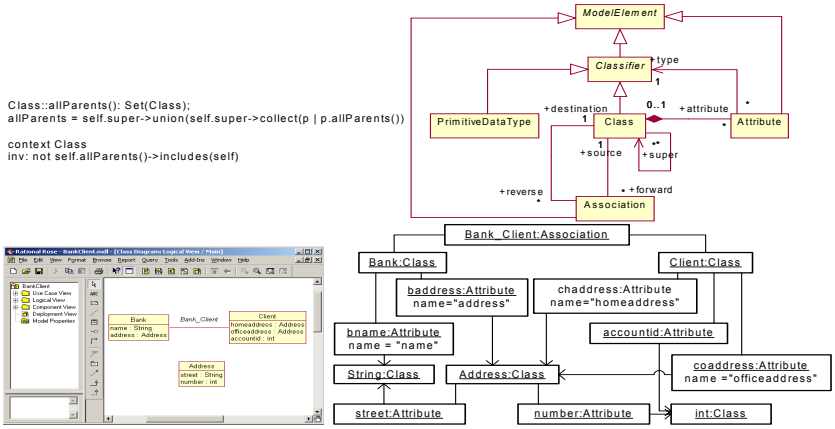



**Fig. 4.** An illustrative example: a model UML conforms to its meta-model

An UML model conforms to the UML meta-model must also satisfied all the well-formedness rules defined with the meta-model. The multiplicity in the meta-model can also be expressed as constraints associated to the meta-model [16]. Furthermore, we have the following principle:

– Every link in the model finds a unique definition in the meta-model. [P3]

This principle is so important as the [P2] principle for a model UML and also for the conformance relation between a model and a meta-model defining it in meta-modeling. These two principles [P2, P3] are also applicable in the "strict meta-modeling" approach in which the OMG's MOF is an example: "Every element of an $M_n$ level model is an *instance_of* exactly one element of an $M_{n+1}$ level model" [1].

## 3 A formalization of the conformance relation in the OOM TS

In a very general way, a model can be viewed as containing:

– A set of model elements (character in a string or regular expression, symbols and terminals in a grammar, element or attribute nodes in XML, model elements in modeling)
– Some of those elements are associated to some sorts of literal (integer, real, string....)
– A set of links that associates elements (link is directed). Those links forms a navigation network among model elements.
– To make sense, each model must be associated with a meta-model defining it.
– Every model element finds its unique definition in the meta-model.
– Every model link finds its unique definition in the meta-model.

The fact that there is a mapping from a model (the defined artifact) and its meta-model (the defining artifact) is one of the *necessary conditions* for the model to conform to its meta-model. This mapping includes model elements mapping and model links mapping and is then a structural mapping. Together with this structural mapping the model must satisfy constraints associated to the meta-model. Those

constraints can be evaluated based on structural mapping and literal values associated to model elements.

Before taking into details of the formalization, we put some words about the category theory. Category theory originally arose in mathematics out of the need of formalism to describe the passage from one type of mathematical structure to another [7]. Category theory has been used in diverse branches of software engineering and computer science as pointed out by Goguen [8], in object-oriented software evolution [11] and recently the formalization of UML [14] and MOF [4] etc. In category theory there are structures called categories that contain objects and morphisms. Those morphisms can be composed and the composition of morphisms is associative. Functor is a structure-preserving mapping between two categories. Definitions of category, functor and other notion of category theory can be found at [15], [7]. A computational aspect of category theory can be found in [12].

The next topic is the proposed formalization of the conformance relation between a model and its meta-model in the OOM TS. The OOM TS bases on OMG's technology (MOF, UML, QVT...), which is originally based on object models. Adapted from [13], an object model is a tuple

$\mu = (\mathtt{CLASS}, \mathtt{ATT}_c, \mathtt{OP}_c, \mathtt{ASSOC}, \mathtt{associates}, \mathtt{roles}, \mathtt{multiplicities}, <, \mathtt{PRIMITIVETYPE})$

such that

i. `CLASS` is a set of classes.
ii. $\mathtt{ATT}_c$ is a set of operation signatures for functions mapping an object of class c to an associated attribute value.
iii. $\mathtt{OP}_c$ is a set of signatures for user-defined operations of a class c.
iv. `ASSOC` is a set of association names.
   a. `associates` is a function mapping each association name to a list of participating classes.
   b. `roles` is a function assigning each end of an association a role name.
   c. `multiplicities` is a function assigning each end of an association a multiplicity specification.
v. $<$ is a partial order on `CLASS` reflecting the generalization hierarchy of classes.
vi. `PRIMITIVETYPE` is a set of primitive data types used in the object model = {`STRING, INTEGER, REAL`}.

In our formalization, model navigation plays an important role. We proposed the concept of *navigation morphism* which is represented by a tuple

$\mathtt{nav} = (e_s, L, E_t)$

such that

i. $e_s$ is the model element that is the source of the navigation morphism
ii. L is a sequence of navigation label
iii. $E_t$ is a sequence of elements that is orderly located in the navigation from the source element $e_s$ to the target element. The last element of this sequence is the target of the navigation morphism.

Now, from every object model μ, there is a derived category $C_\mu$ :

$C_{\mu} = (Ob_c, Mor_c, dom, cod, id, composition)$

such that

i. $Ob_c = CLASS \cup PRIMITIVETYPE$

ii. PRIMITIVETYPE is the set of primitive types used in the object model

iii. $Mor_c = Mor_{C1} \cup Mor_{C2}$

iv. $Mor_{C1}$ is the set of all navigation morphisms $(e_s, [role\ name], [e_t])$ representing a navigation from $e_s$ to $e_t$ ($e_s, e_t \in CLASS$) through the "role name" role. $Mor_{C1}$ can be calculated from CLASS, ASSOC, associates and roles.

v. $Mor_{C2}$ is the set of all navigation morphisms $(e_s, [attribute\ name], [e_t])$ representing a navigation from $e_s$ ($e_s \in CLASS$) to $e_t$ ($e_s \in$ PRIMIVITES) through the "attribute name" attribute. $Mor_{C1}$ can be calculated from CLASS, $ATT_c$, PRIMITIVETYPE.

vi. dom: $Mor_c \rightarrow Ob_c$ is a function that takes a navigation morphism as argument and gives the source of that navigation morphism as result. This function can be calculated from CLASS, $ATT_c$, ASSOC, associates, roles and <.

vii. cod: $Mor_c \rightarrow Ob_c$ is a function that takes a navigation morphism as argument and gives the target of that navigation morphism as result. This function can be calculated from CLASS, $ATT_c$, ASSOC, associates, roles and <.

viii. id is an identity function that takes a model element e as its argument and give a navigation morphism (e,[],[e]) as result. i.e this function returns a navigation morphism from the element e to itself (there is no navigation label)

ix. composition is a function that takes two navigation morphisms nmor1 = $(e_{s1}, L_1, E_{t1})$ and nmor2 = $(e_{s2}, L_2, E_{t2})$ as its arguments and give a composite navigation morphism nmor=$(e_{s1}, L_1$ concat $L_2, E_{t1}$ concat $E_{t1})$ when cod(nmor1)=dom(nmor2)

Once the model μ is promoted as a meta-model ($M_2$ level), any model of this meta-model can be represented by a category :

$C_{model} = (Ob_c, Mor_c, dom, cod, id, composition)$

such that

i. $Ob_c = OBJECT \cup LITERAL$

ii. OBJECT is the set of objects in the selected model

iii. LITERAL is the set of objects associated to a primitive value used in the selected model

iv. $Mor_c = Mor_{C1} \cup Mor_{C2}$

v. $Mor_{C1}$ is the set of all navigation morphisms $(e_s, [role\ name], [e_t])$ representing a navigation from $e_s$ to $e_t$ ($e_s, e_t \in$ OBJECT) through the "role name" role. $Mor_{C1}$ can be calculated from the selected model.

vi. $Mor_{C2}$ is the set of all navigation morphisms
$(e_s, [attribute\ name], [e_t])$
representing a navigation from $e_s$ ($e_s \in OBJECT$) to $e_t$ ($e_s \in LITERAL$) through the "attribute name" attribute. $Mor_{C1}$ can be calculated from the selected model.

vii. $dom: Mor_C \rightarrow Ob_c$ is a function that takes a navigation morphism as argument and gives the source of that navigation morphism as result. This function can be calculated from the selected model.

viii. $cod: Mor_C \rightarrow Ob_c$ is a function that takes a navigation morphism as argument and gives the target of that navigation morphism as result. This function can be calculated from the selected model.

ix. $id$ is an identity function that takes a model element $e$ as its argument and give a navigation morphism $(e,[],[e])$ as result. i.e this function returns a navigation morphism from the element $e$ to itself (there is no navigation label)

x. $composition$ is a function that takes two navigation morphisms $nmor1 = (e_{s1}, L_1, E_{t1})$ and $nmor2 = (e_{s2}, L_2, E_{t2})$ as its arguments and give a composite navigation morphism
$nmor = (e_{s1}, L_1\ concat\ L_2, E_{t1}\ concat\ E_{t1})$
when $cod(nmor1) = dom(nmor2)$

**An example: BankClient model conforms to SimpleUML model**

The simplified meta-model UML and the Bank_Client model (Fig.4) are illustrated partially in the categorical form in the Fig. 5. Model elements and model links of these two models is provided in the Table.1.

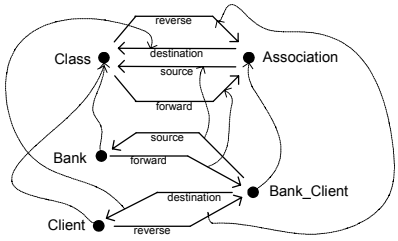



**Fig. 5.** A partial view of mapping from BankClient (model) to SimpleUML (meta-model)

The mapping from Bank_Client model to SimpleUML model illustrated in the Table.2 can be expressed by a functor $F: C_{Bank\_Client} \rightarrow C_{SimpleUML}$ that contains:

– A model element mapping
$F_{element} = Bank \rightarrow Class\ ;\ Client \rightarrow Class\ ;\ Bank\_Client \rightarrow Association$

– A model link mapping $F_{navigation} =$
*(Bank,[forward],[Bank_Client])* $\rightarrow$ *{(Class,[forward],[Association]) ;*
*(Client,[reverse],[Bank_Client])* $\rightarrow$ *(Class,[reverse],[Association]) ;*
*(Bank_Client,[source],[Bank])* $\rightarrow$ *(Association,[source],[Class]) ;*
*(Bank_Client,[destination],[Client])* $\rightarrow$ *(Association,[destination],[Class])*

**Table 1.** Model elements and model links of Bank_Client and SimpleUML model

| | $C_{Bank\_Client}$ | $C_{SimpleUML}$ |
|---|---|---|
| elements | *{Bank,Client,Bank_Client}* | *{Class, Association}* |
| links/ basic navigations | *{(Bank,[forward],[Bank_Client]), (Client,[reverse],[Bank_Client]), (Bank_Client,[source],[Bank]), (Bank_Client,[destination],[Client])}* | *{(Class,[forward],[Association]), (Class,[reverse],[Association]), (Association,[source],[Class]), (Association,[destination],[Class])}* |

**Table 2.** Navigation mapping and mapping of a composition

| From Bank_Client | To SimpleUML |
|---|---|
| *(Bank,[forward],[Bank_Client])* | *(Class,[forward],[Association])* |
| *(Bank_Client,[destination],[Client])* | *(Association,[destination],[Class])* |
| *(Bank,[forward],[Bank_Client]) ° (Bank_Client,[destination],[Client])= (Bank,[forward,destination],[Bank_Client,Client])* | *(Class,[forward],[Association]) ° (Association,[destination],[Class])= (Class,[forward,destination],[Association,Class])* |

**Remarks**. The mapping of the composition of two navigations is the composition of the mappings of the two navigations. This is an important property of the structural mapping and is called structure-preserving mapping in the category theory.

## 4 Exploiting the formalization

In order to demonstrate the benefits of the proposed formalization, we have developed a prototype of an MDE environment in which different kind of data such as models, meta-models, mapping specifications, conformance relationships and more generally, any structure-preserving relationship can be represented in a unified manner (using categories and functors).

The developed prototype having architecture depicted in Fig.6 contains an OCL evaluator that exploits categorical representations of models and conformance mapping to navigate through model elements. The implementation of this evaluator is well facilitated since model navigation – an important part of the language is made explicit in the categorical representation of (meta-)models.

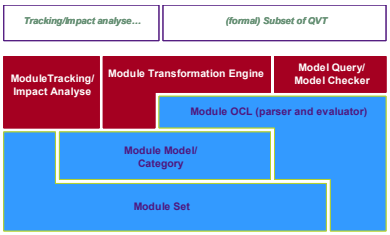



**Fig. 6.** The MDE environment prototype

The developed prototype has allowed us to point out several potential usages of the formalization presented in the previous sections. Some of these usages are provided below:

- *Verifying for model conformance*: the input and output model of a transformation can be respectively verified if each model conforms to its meta-model due to the OCL evaluator.
- *Model query*: models can be queried with the OCL language.

- *Model transformation execution*: a set of model transformations (structure preserving transformation) can be executed due to the transformation engine.
- *Systematic traceability*: the traceability information is stored as categorical functors and is produced as explicit result of transformation together with output model.
- *Tracking for multi-step transformations*: since traceability information is stored in the form of functors, those functors can be composed in the case of successive transformation.
- *Help to the analysis of impacts:* since the structural relation between input and output model is captured by a functor (this functor is also the traceability information), it is possible to ask some kind of questions about transformation executed such as: *if a model element (or model link) in the input model is removed then which parts of the output model will change? Or in the inverse direction: if I want to make some change in the output model, which parts of the input model need to be changed?* These kind of questions can be answered without making real change and re-execute transformations and is very useful in an interactive environment where model transformation is an interactive computer aided tool to the development or may be in the specification phase of model transformation when debugging facility is a requirement.
- *Analysis for (structural) completeness of model transformations:* with the traceability information we can easily verify which parts of the input model do not take part in the generation of any model element in the output model, this *may* be the case in that the specification of model transformation is not complete.

## 5 Related works

Category theory has been used to formalize UML [14] and recently MOF [4]. These formalizations based on Slang, a language supporting category theory of the Kestrel Institute [14]. Our formalization uses directly the graph representation (interpreted as categories) of models, functors to describe conformance mapping and OCL to describe constraints. In our work, functor is also used to represent relation between models at different levels of abstraction of the same system.

## 6 Conclusions

The work presented in this paper bases on a categorical abstraction of model and OCL to formalize the conformance relation of a model to its meta-model in the Object-Oriented Modeling TS. This relation can be expressed by a conformance mapping from the model to its meta-model and a set of constraints associated to the meta-model. These constraints must be satisfied when being evaluated over the model, the meta-model and the conformance mapping between them. We believe that the same kind of formalization can be used to other TSs due to the conformance mapping from a model to its definition (meta-model) in OOM TS or from a XML document to its DTD (or XML Schema), etc. The main advantage of this formalization is that it is

very abstract and can be applied to any kind of (meta-)models. This formalization is also a first step in defining a model transformation formalism in which traceability and analysis of impacts is fully supported.

## References


1. Colin Atkinson. Meta-Modeling for Distributed Object Environments. In *The First International Enterprise Distributed Object Computing Conference (EDOC '97)* , pages 90-103, Brisbane, Australia, October 1997. IEEE Computer Society Press.
2. Jean Bézivin. On the Basic Principles of Model Driven Engineering. In *MDE for Embedded System Summer School*, Brest, France, September 2004. ENSIETA.
3. Jean Bézivin. On the unification power of models. *SoSym*, 2005. [http://www.sciences.univ-nantes.fr/lina/atl /www/papers/OnTheUnificationPowerOfModels.pdf]
4. Kenneth Baclawski, Mieczyslaw Kokar, and Jeffrey Smith. Metamodeling facilities. [ http://www1.coe.neu.edu/%7Ejsmith/Publications/mof.pdf ]
5. Denilson Barbosa, Alberto O. Mendelzon, Leonid Libkin, Laurent Mignet, and Marcelo Arenas. Efficient Incremental Validation of XML Documents. In *ICDE*, 2004. [ http://www.cs.toronto.edu/~marenas/publications/icde04.pdf ]
6. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. *Extensible Markup Language (XML) 1.0 (Third Edition)*. W3C, February 2004. [ http://www.w3.org/TR/2004/REC-xml-20040204/ ]
7. Michael Barr and Charles Wells. Category Theory - Lecture Notes for ESSLLI. Lecture Notes, 1999. [ http://www.folli.uva.nl/CD/1999/library/pdf/barrwells.pdf ]
8. Joseph A. Goguen. A Categorical Manifesto. *Mathematical Structures in Computer Science*, 1(1):49-67, 1991. [ http://citeseer.ist.psu.edu/goguen91categorical.html ]
9. Kathleen Jensen and Niklaus Wirth. *Pascal User Manual and Report*. Springer-Verlag, 1976.
10. I. Kurtev, J. Bézivin, and M. Aksit. Technical spaces: An initial appraisal. In *CoopIS, DOA 2002 Federated Conferences*, Irvine, 2002. [http://www.sciences.univ-nantes.fr/lina/atl/www/papers/PositionPaperKurtev.pdf]
11 Tom Mens. *A Formal Foundation For Object-Oriented Software Evolution*. PhD thesis, Vrije Universiteit Brussel, August 1999.
12. David E. Rydeheard and Rod M. Burstall. *Computational Category Theory*. Series in Computer Science. Prentice Hall International, 1988. [ http://www.cs.man.ac.uk/~david/categories/book/book.pdf]
13. Mark Richters. *A Precise Approach to Validating UML Models and OCL Constraints*. PhD thesis, Universität Bremen, 2002. [ http://www.db.informatik.uni-bremen.de/teaching/courses/ss2002_oose/m.pdf]
14. Jeffrey E. Smith. *UML Formalisation and Transformation*. PhD thesis, Northeastern University, Boston, Massachusetts, December 1999.
15. Jaap van Oosten. Basic Category Theory. In *Basic Research in Computer Science*, BRICS Lecture Series. University of Aarhus, January 1995. [ http://www.brics.dk/LS/95/1/BRICS-LS-95-1/BRICS-LS-95-1.html]
16. Jos Warmer and Anneke Keleppe. *The Object Constraint Language, Precise Modeling With UML*. Object Technology Series. Addison-Wesley, 1999.

# Dependencies between Models in the Model-driven Design of Distributed Applications[1]

João Paulo A. Almeida, Luís Ferreira Pires, Marten van Sinderen

Centre for Telematics and Information Technology, University of Twente
PO Box 217, 7500 AE Enschede, The Netherlands
{almeida, pires, sinderen}@cs.utwente.nl

**Abstract.**. In our previous work, we have defined a model-driven design approach based on the organization of models of a distributed application according to different levels of platform-independence. In our approach, the design process is structured into a preparation and an execution phase. In the preparation phase, (abstract) platforms and transformation specifications are defined. These results are used by a designer in the execution phase to develop a specific application. In this paper, we analyse the dependencies between the various types of models used in our design approach, including platform-independent and platform-specific models of the application, abstract platforms, transformation specifications and transformation parameter values. We consider models as modules and employ a technique to visualize modularity which uses Design Structure Matrices (DSMs). This analysis leads to requirements for the various types of models and directives for the design process which reduce undesirable dependencies between models.

## 1 Introduction

In our previous work [1, 2], we have defined a model-driven design approach (aligned with the Model-Driven-Architecture [7]) based on the organization of models of a distributed application according to different levels of platform-independence. In this approach, models at a particular level of platform-independence can be realized with a number of platforms (such as, e.g., middleware platforms), possibly through application of successive (automated) transformations that lead ultimately to platform-specific models, i.e., models at the lowest level of platform-independence with respect to a particular definition of platform.

An important architectural concept of our approach is that of an abstract platform. An abstract platform is an abstraction of infrastructure characteristics assumed for models of an application at a certain level of platform-independence. An abstract platform is represented through metamodels, profiles and reusable design artefacts [1]. For example, if a platform-independent design contains application parts that interact through operation invocations (e.g., in UML [8]), then operation invocation is a characteristic of the abstract platform. Capabilities of a concrete platform are used

[1] This work is part of the Freeband A-MUSE project. Freeband (http://www.freeband.nl) is sponsored by the Dutch government under contract BSIK 03025.

during platform-specific realization to support this characteristic of the abstract platform. For example, if CORBA is selected as a target platform, this characteristic can be mapped onto CORBA operation invocations.

An indispensable activity in early stages of our development approach is to determine the levels of models, the abstract platforms, and the (automated) transformations that are needed. This activity is part of the *preparation phase* of the MDA development process [6]. In the preparation phase, (MDA) experts define the metamodels, profiles and transformations that are to be used in the *execution phase* by application developers. In the execution phase, a specific application is developed using the generalized designs and design knowledge captured during the preparation phase.

Figure 1 shows the various models manipulated in our approach. Three levels of platform-independence are depicted, and the results are classified according to the phase in which they are produced. In this figure, an arrow indicates that a model is dependent on the existence of another model by construction. Abstract platforms have been depicted as models, indicating that abstract platform definitions can be captured in *abstract platform models*. Transformation specifications have also been depicted as models, indicating that generalized design operations can be captured and reused. Transformation specifications can be parameterized and values for transformation parameters are defined in the execution phase. These values are called transformation arguments. Arguments of a transformation are also called *markings* when these are associated to elements in a source model, in which case transformation parameters are called *marks*.

Ideally, models in our approach (presented in Figure 1) should be independent of each other, i.e., it should be possible to create models independently, and a modification in one model should not impact other models. Nevertheless, models capture design decisions on the same object of design, i.e., the same application, and hence not all models are independent of each other. The benefits of separation of models are reduced when models are related in such a way that modifications in a model affect other models. In this paper, we analyse the dependencies between the various types of models used in our design approach and strive to find techniques to avoid undesirable dependencies between models.

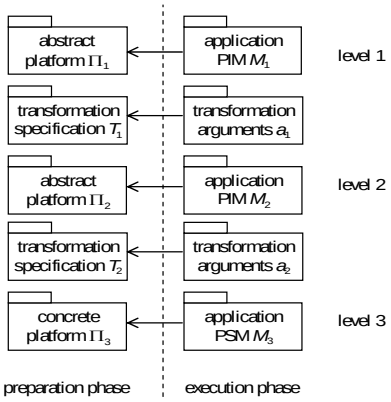



**Fig. 1.** Models in our design approach

Dependencies between models restrict the opportunities for division of labour and concurrent design. Interdependencies reduce the efficiency of the design process and often have to be addressed in the design process by introducing iteration cycles [4]. As we elaborate in this paper, some interdependencies can be avoided by following a number of rules with respect to the content of the various models and with respect to the modifications that may be applied to the various models.

In the remainder of this paper, we address the following questions with respect to the separation of models in our approach (among others):

- can concrete platforms be modified without affecting PIMs and abstract platforms?
- can transformation specifications be modified without affecting PIMs and abstract platforms?
- does a modification in a PIM affects a corresponding PSM?
- does a modification in a PSM affects a corresponding PIM?
- are there interdependencies between the various models that require iterations in the design process? Can these be avoided?

This paper is further organised as follows: section 2 proposes that models should be considered as modules whose modularity can be analysed through a technique called Design Structure Matrices (DSMs) [9, 10]; section 3 analyses the (inter)dependencies between the various types of models, which results in requirements and guidelines for the separation of models; section 4 discusses how the dependencies between models affect the design process; section 5 classifies the different models according to their various dependencies; finally, section 6 presents some concluding remarks.

## 2 Models as modules

In order to examine the relations between the various models, we consider models as *modules*. Typically, a module is a set of elements of a design that are grouped together according to an architecture or plan, with three main purposes [3, 4]: to make complexity manageable; to enable parallel work; and to accommodate future uncertainty.

While modularization is often used as a technique to split up and assign different functions of a complex system to different system parts, we split up and assign different design decisions to different models. A number of basic principles of modularity apply both to the functional decomposition of system parts (within a model) and to the separation of models in our design approach.

As is noted in [4]: “a complex engineering system is modular-in-design if (an only if) the *process of designing it can be split up and distributed across different separate modules* that are coordinated by design rules, not by ongoing consultations amongst the designers.” This definition reveals two important features of systems that are modular-in-design:

- *Independence*: The absence of ongoing consultations amongst the designers of different modules reveals that modules should be largely independent of each other. Modules correspond to independent activities in the design process; and

– *Dependence*: The relations between the different modules are defined by a set of design rules[2] to be respected. These design rules reflect the need for coordination of design choices. Separating strongly related modules forces the number of design rules to increase, constraining the freedom of designers of the different modules.

In the following sections, we examine independence and dependence of models in our design approach. We employ a technique to visualize modularity-in-design which uses Design Structure Matrices (DSMs) [9, 10]. DSMs have been used extensively in the field of Engineering Design, both for products and production processes and design processes [4]. In this technique, modules are arrayed along the rows and columns of a square matrix. The matrix is filled in by determining, for each module, which other modules affect it and which are affected by it. The result is a map of the dependencies between the modules.

## 3 Dependencies between models: two levels of models

We start our analysis by assuming two levels of design within a single design iteration cycle as depicted within the rounded rectangle in Figure 2.

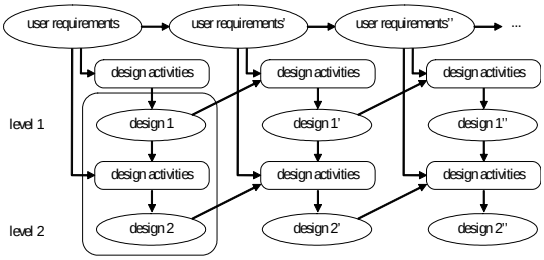



**Fig. 2.** Two levels of models related by transformation

We assume further that the preparation phase results in an abstract platform $\Pi_1$ for designs at level 1, a concrete platform $\Pi_2$ for designs at level 2. The design activities are constrained by a transformation specification $T_1$ that relates models that rely on $\Pi_1$ to models that rely on $\Pi_2$. This situation is depicted in Figure 3. This figure reveals the various models of the execution phase that are considered at this point of our analysis, namely, an application PIM, transformation arguments, and an application PSM.

[2] In functional decomposition, interfaces between components are considered design rules.

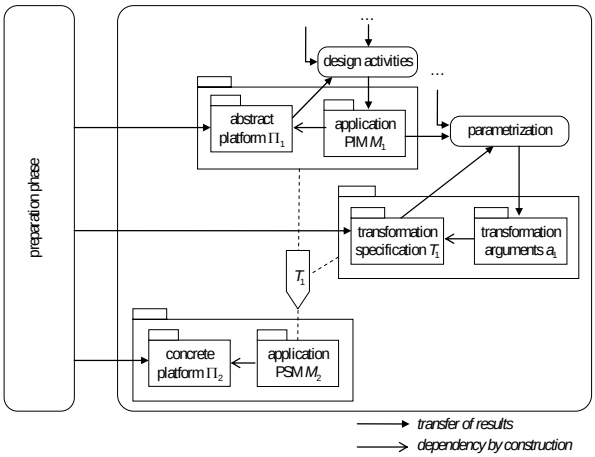



**Fig. 3.** Two levels of models related by transformation

We discuss the dependencies between each of the models depicted in Figure 3 in the following sections. In each section, we discuss how the various models are affected as a result of a modification of one of the other models. After the relations between all models are examined, a DSM is built to visualize the dependencies between the various models.

**Application PIM.** Table 1 shows the dependencies between the various models and an application PIM. The '⊠' symbol marks the existence of some dependency. The absence of the symbol indicates there is no dependency. We justify the existence or absence of a dependency for each pair of models.

**Table 1.** Dependencies between the various models and an application PIM

| | Application PIM | Explanation |
|---|---|---|
| Application PIM | N/A | trivial |
| Abstract platform | | An abstract platform is designed so that it can be used to design a class of applications; the modified application PIM is still a member of this class of applications.<br>This constitutes a generality requirement for abstract platform, but also sets the constraints on possible modifications of an application PIM for a given abstract platform. |
| Application PSM | ☒ through transformation | The relations between application PIMs and PSMs are determined by transformation specifications and transformation arguments; if the application PIM is modified, it is possible that the modified PIM and the original PSM no longer respect this relation; in this case, the PSM or transformation arguments may be affected by change. |
| Concrete platform | | The concrete platform is a member of the set of target platforms implied by portability requirements; all application PIMs that rely on the abstract platform must be buildable (see explanation below about buildability) in the concrete platform, thus requiring no modifications in the concrete platform.<br>This constitutes requirements for the abstract platform and transformation specification. |
| Transf. arguments | ☒ | Transformation arguments are used to introduce variation in transformation specifications, in order to capture particular design decisions; these decisions may be application-specific or may refer to elements of the application PIM; e.g., transformation parameters can be used to specify the physical allocation of each application component in the application PIM. |
| Transf. specification | | Transformation specifications are designed so that they can be applied to the class of applications that can be built on top of an abstract platform; the modified PIM is still a member of this class of applications.<br>This constitutes a generality requirement for transformation specification. |

*Buildability* of a design is inversely proportional to the amount of time, effort and resources required to build a conformant realization of the design on *a particular platform*. Buildability depends on the contents of a design. The actual contents of a platform-independent design depend partly on the abstract platform, which is defined in the preparation phase. Therefore, in the preparation phase, buildability can only be estimated indirectly, by analysing the impact of abstract platform characteristics in the buildability of the class of application designs supported by the abstract platform. We propose this is done by examining the differences and similarities in the abstract platform and target platforms[3].

Having introduced the notion of buildability, we are able to formulate a definition of platform-independence of a design. We say that a design is *platform-independent* if, and only if, it is buildable on a number of target platforms. The set of target platforms is determined by *portability requirements* for the design, which are themselves determined by technical, business and strategic arguments.

**Abstract platform.** Table 2 shows the dependencies between the various models and an abstract platform.

[3] We have explored this idea initially in [2].

**Table 2.** Dependencies between the various models and an abstract platform

| | Abstract platform | Explanation |
|---|---|---|
| Application PIM | ☒ | By definition: “an abstract platform is an abstraction of infrastructure characteristics assumed in the construction of PIMs of an application”; if these characteristics change, the application PIM may be affected. |
| Abstract platform | N/A | trivial |
| Application PSM | | Modifying an abstract platform may affect PIMs, transformation specifications (see respective cells in this table), which in turn may affect application PSMs (see other tables); however, only direct dependencies are represented in a DSM. |
| Concrete platform | | The set of target platforms is determined by portability requirements; during abstract platform definition, buildability with respect to the target platform must be observed.<br>This constitutes a requirement for abstract platform definition. |
| Transf. arguments | | Transformation arguments depend on the transformation specification, which depends on abstract platforms (see cell below); however, only direct dependencies are represented. |
| Transf. specification | ☒ | The abstract platform defines the common characteristics of a class of platform-independent designs for which there should be generalized implementation relations to different platforms; these implementation relations are captured in transformation specifications; a change in abstract platform characteristics changes the class of applications, invalidating assumptions on common concepts, patterns and structures that were made to define transformations. |

The separation between an abstract platform and a transformation specification is analogous to the separation between an interface definition and a realization of the interface in component-based design: an abstract platform defines requirements which are satisfied by one or several transformation specifications.

**Application PSM.** Table 3 shows the dependencies between the various models and an application PSM.

**Table 3.** Dependencies between the various models and an application PSM

| | Application PSM | Explanation |
|---|---|---|
| Application PIM | ☒ through transformation | The relations between application PIMs and application PSMs are determined by transformation specifications and transformation arguments; if the application PSM is modified, it is possible that the modified PSM and the original PIM no longer respect this relation; in this case, the PIM or transformation arguments may be affected by change. This dependency exists for both unidirectional and bidirectional [5] transformations. In the case of bidirectional transformations, changes to PIM may be propagated automatically. |
| Abstract platform | | A modification in an application PSM may result in a modification in the application PIM (see cell *application PIM* above); the modified PIM is still a member of this class of applications for which the abstract platform is defined.<br>This constitutes a generality requirement for abstract platform, but also sets the constraints on modifications of an application PSM for a given abstract platform. |
| Application PSM | N/A | trivial |
| Concrete platform | | A concrete platform is designed so that is can be used to design a class of applications; the modified PSM is still a member of this class of applications.<br>This constitutes a generality requirement for concrete platforms. |
| Transf. arguments | ☒ through transformation | (see cell *application PIM* above) |
| Transf. specification | | Transformation specifications define generalized implementation relations; transformation specifications define a class of PSMs that conform with PIMs; the modified PSM is still a member of this class of applications.<br>This constitutes a generality requirement for transformation specifications, but also sets the constraints on possible modifications of an application PSM for a given transformation specification and a PIM. |

**Concrete platform.** Table 4 shows the dependencies between the various models and a concrete platform.

**Table 4.** Dependencies between the various models and a concrete platform

| | Concrete platform | Explanation |
|---|---|---|
| Application PIM | independence is engineered | Independence is engineered in the definition of abstract platforms. This constitutes a buildability requirement for abstract platforms. |
| Abstract platform | independence is engineered | Independence is engineered in the definition of abstract platforms. This constitutes a buildability requirement for abstract platforms. |
| Application PSM | ☒ | Application PSM depends on sets of concepts, patterns and structures provided by a concrete platform; the instability of concrete platforms, and hence application PSMs, motivates separation of platform-independent and platform-specific concerns in our approach. |
| Concrete platform | N/A | trivial |
| Transf. arguments | ☒ | Transformation arguments may be platform-specific, e.g., markings may define that particular components should be transformed into Session or Message-Driven Enterprise Java Beans. |
| Transf. specification | ☒ | Transformation specifications define generalized implementation relations for a particular target platform; change the target platform and these relations may be invalidated. Ideally, this dependency could be reduced by using concrete platform models as transformation arguments. However, this solution requires highly general transformation specifications, which define generalized implementation relations for a class of target platforms (resulting in a platform-independent transformation specification). |

**Transformation arguments.** Table 5 shows the dependencies between the various models and transformation arguments.

**Table 5.** Dependencies between the various models and transformation arguments

| | Transf. arguments | Explanation |
|---|---|---|
| Application PIM | | Abstract platforms are defined to preserve freedom of implementation, so that different implementations of application PIMs built on top of it are possible; since transformation arguments are used to introduce variations in generalized implementation relations, changes in transformation arguments should not affect application PIMs nor abstract platforms.<br>This constitutes a requirement for abstract platforms and transformations, and sets the constraints on possible modifications of transformation arguments for a given combination of abstract platform and transformation specification. |
| Abstract platform | | (see cell *application PIM* above) |
| Application PSM | ☒ through transformation | The relations between PIMs, transformation arguments and PSMs are determined by transformation specifications; if transformation arguments are modified, it is possible that the original PIM, the modified arguments and the original PSM no longer respect this relation; in this case, the PSM may be affected by change in transformation arguments. |
| Concrete platform | | A concrete platform is designed so that is can support a class of applications; a PSM that is affected by a change in transformation arguments is still a member of this class of supported applications, therefore, requiring no modification of the concrete platform.<br>This constitutes a requirement for transformation specification, namely that the results of transformations are always PSMs that use the concrete platform. |
| Transf. arguments | N/A | trivial |
| Transf. specification | | Transformation specifications have transformation parameters, which are assigned values when the transformation specification is instantiated. |

From the perspective of model transformation, the distinction between PIMs and transformation arguments is unnecessary: both PIMs and transformation arguments may be considered as input information for an unparameterized transformation. However, the distinction is relevant from the perspective of the design process: PIMs are *platform*- and *transformation independent*, while transformation arguments may be *platform*- and *transformation specific*. Transformation arguments may be defined after PIMs have been conceived. As a consequence, designers of PIMs may not be aware of whatever transformation parameters may be chosen by a designer using the PIM as a starting point to derive a PSM.

**Transformation specification.** Finally, Table 6 shows the dependencies between the various models and a transformation specification.

**Table 6.** Dependencies between the various models and a transformation specification

| | Transf. specification | Explanation |
|---|---|---|
| Application PIM | | Abstract platforms are defined to preserve freedom of implementation, so that different implementations of application PIMs built on top of it are possible; these different implementations are captured in transformation specifications.<br>This constitutes a requirement for abstract platform, but also sets the constraints on possible modifications of transformation specifications for a given abstract platform. |
| Abstract platform | | (see cell *application PIM* above) |
| Application PSM | ☒ | The relation between application PIM and application PSM is determined by transformation specifications and transformation arguments; since a change in transformation specification should not affect PIMs (see cell *application PIM* above), modifications to transformation specifications must be accommodated in the PSM or in transformation arguments. |
| Concrete platform | | PSMs related by transformation specifications must be realizable on top of a concrete platform.<br>This constitutes a requirement for transformation specifications. |
| Transf. arguments | ☒ | Transformation parameters are used to introduce variations in generalized implementation specifications; if a transformation specification is modified parameters may be modified and new parameters may be introduced, affecting transformation arguments. |
| Transf. specification | N/A | trivial |

Since transformation arguments may be transformation-specific, transformation arguments must be captured separately from PIMs so that PIMs do not become transformation-specific. Therefore, in case of parameterization by marking, the unmarked PIM must be kept separately from markings. The unmarked PIM and markings can be combined into a marked model for the purposes of transformation if necessary.

**Design Structure Matrix.** Table 7 provides an overview of the dependencies between each of the models considered in our analysis so far. The columns of this table correspond to the columns of tables 1 to 6. When the table is read row-wise, the '⊠' mark indicates that the model that names to the row is affected by the models that name each of the columns. When the table is read column-wise, the mark shows the models that may be affected directly as a result of a modification in the model that names the column.

**Table 7.** Dependencies between models: Design Structure Matrix

| | Application PIM | Abstract platform | Application PSM | Concrete platform | Transf. arguments | Transf. specification |
|---|---|---|---|---|---|---|
| Application PIM | N/A | ⊠ | ⊠ through transformation | independence is engineered | | |
| Abstract platform | | N/A | | independence is engineered | | |
| Application PSM | ⊠ through transformation | | N/A | ⊠ | ⊠ through transformation | ⊠ |
| Concrete platform | | | | N/A | | |
| Transf. arguments | ⊠ | | ⊠ through transformation | ⊠ | N/A | ⊠ |
| Transf. specification | | ⊠ | | ⊠ | | N/A |

DSMs exhibit an interesting property for our analysis: if we consider that there is a time sequence associated with the position of the elements in the matrix, then all marks above the diagonal are considered feedback marks [11]. Feedback marks require iterations in the sequence of tasks executed. DSMs can be manipulated to eliminate or reduce feedback marks, e.g., by reordering the sequence of elements in the matrix. It is also possible to group elements of the matrix into clusters, a technique which allows us to consider the set of elements of a cluster as a single module.

In the following section, we manipulate the DSM represented in Table 7 to show how the dependencies between models affect the design process.

## 4 Dependencies between models and the design process

**Preparation and execution phase concerns.** Table 8 shows a reordered DSM. The models that result from the preparation activities, namely, concrete and abstract platforms and transformation specifications are placed in the first three positions of the matrix. These models are grouped into a cluster, which represents the preparation phase. A second cluster represents the execution phase, grouping application PIM, transformation arguments and application PSM.

**Table 8.** Clustering dependencies with respect to preparation and execution activities

| | Concrete platform | Abstract platform | Transf. specification | Application PIM | Transf. arguments | Application PSM |
|---|---|---|---|---|---|---|
| Concrete platform | N/A | | | | | |
| Abstract platform | independence is engineered | N/A | | | | |
| Transf. specification | ☒ | ☒ | N/A | | | |
| Application PIM | independence is engineered | ☒ | | N/A | | ☒ through transformation |
| Transf. arguments | ☒ | | ☒ | ☒ | N/A | ☒ through transformation |
| Application PSM | ☒ | | ☒ | ☒ through transformation | ☒ through transformation | N/A |

The absence of feedback marks above the diagonal formed by the preparation and execution phase clusters in Table 8 shows that the preparation phase does not depend on the execution phase. This result is made possible by requirements imposed on the preparation phase. These requirements are described in the cells of tables 1 to 6 that correspond to the cells positioned above the diagonal formed by the two clusters. Failure to satisfy these requirements would imply the presence of feedback dependencies, which would require revisiting the preparation phase. The absence of feedback marks above the diagonal formed by the preparation and execution phase clusters can be summarized by the following design rule:

*Changes in PIM, PSM or transformation arguments must be accommodated in PIM, PSM or transformation arguments, but not in the abstract platform, concrete platform nor transformation specification.*

Table 8 also reveals the absence of feedback dependencies within the preparation phase, since, within the cluster, no feedback marks appear above the diagonal. The same, however, cannot be said of the execution phase: modifications in the application PSM may affect the PIM and transformation arguments. The presence of feedback dependencies in the execution phase is addressed through iteration in the execution phase. An iteration in the execution phase allows a designer to gain insight into the implications of design decisions at the PIM-level for the application PSM, which may result in adjusting the PIM in a subsequent iteration.

However, for the design process to advance towards a stable application PIM, it is necessary that the dependencies between PSM and PIM should eventually decrease. Eventually, the application PIM must be such that it does not depend on design decisions that constrain the choice of target platform. This constitutes an important requirement for the iterative approach in the execution phase.

**Multiple levels of models.** We continue our analysis by considering the dependencies between the models at three different levels related by transformation. Table 9 shows the dependencies between the various models. These dependencies are clustered for each pair of consecutive levels of models, i.e., a cluster for models of levels 1 and 2 and a cluster for models of levels 2 and 3. This DSM is build by reapplying the transformation pattern, which explains the isomorphic nature of the dependencies in the two clusters.

**Table 9.** Clustering dependencies with respect to levels of models

| | Abstract platform $\Pi_1$ | Application PIM $M_1$ | Transf. specification $T$ | Transf. arguments $a_1$ | Abstract platform $\Pi_2$ | Application PIM $M_2$ | Transf. specification $T$ | Transf. arguments $a_2$ | Concrete platform $\Pi_3$ | Application PSM $M_3$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Abstract platform $\Pi_1$ | N/A | | | | | | | | | |
| Application PIM $M_1$ | ⊠ | N/A | | | | ⊠ | | | | |
| Transf. specification $T_1$ | ⊠ | | N/A | | ⊠ | | | | | |
| Transf. arguments $a_1$ | | ⊠ | ⊠ | N/A | ⊠ | ⊠ | | | | |
| Abstract platform $\Pi_2$ | | | | | N/A | | | | | |
| Application PIM $M_2$ | | ⊠ | ⊠ | ⊠ | ⊠ | N/A | | | | ⊠ |
| Transf. specification $T_2$ | | | | | ⊠ | | N/A | | ⊠ | |
| Transf. arguments $a_2$ | | | | | | ⊠ | ⊠ | N/A | ⊠ | ⊠ |
| Concrete platform $\Pi_3$ | | | | | | | | | N/A | |
| Application PSM $M_3$ | | | | | | ⊠ | ⊠ | ⊠ | ⊠ | N/A |

The table shows an overlap between the two clusters. This overlap indicates that the design activities in the different levels are not completely independent, and that the intermediate model PIM forms the ‘interface’ between the two clusters, as could be expected.

## 5 Classifications of models

This section concludes our analysis by classifying the various models and design decisions according to the following dimensions of separation of separation of concerns:

- platform-independent and platform-specific concerns;
- application-independent and application-specific concerns, which correspond to preparation and execution phases concerns, respectively; and,
- transformation-independent and transformation-specific concerns.

Figure 4 places the different models according to the first two dimensions. Three levels of models are depicted.

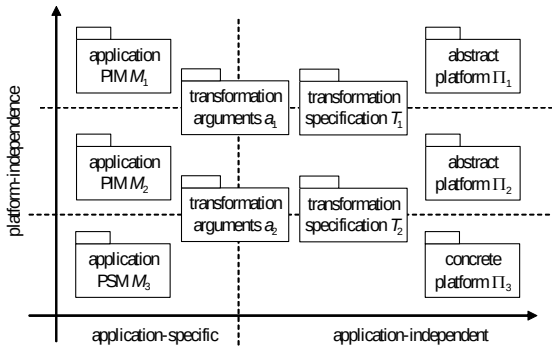



**Fig. 4.** Dimensions of separation of concerns and models

In Figure 4, transformation specifications are placed in the boundary between two levels of platform-independence. This is to denote that transformation specifications rely on the (abstract) platforms of both source and target levels of models (see Table 2 and Table 4). In addition, transformation specifications may also capture some transformation rules which are independent of the target platform.

Similarly to transformation specifications, transformation arguments are also placed in the boundary between two levels of platform-independence. In addition, transformation arguments are placed in the boundary between the application-specific and application-independent concerns area. This is to denote that arguments may be application-specific (see Table 1), but may also capture application-independent design decisions. Application-specific transformation parameterization is used to improve the generality of transformation specifications with respect to specific applications. Application-independent transformation parameterization is used to improve flexibility of transformation specifications in general, e.g., to cope with to variation in user requirements that are not captured in the source models but that are to be addressed during transformation. An example of an application-independent transformation argument determines that, irrespective of the application model, all application parts should be allocated to the same unit of deployment of the target platform.

In addition to the dimensions considered in Figure 4, we can also classify models related in a transformation step as *transformation-independent* or *transformation-specific*. This classification is relative to a transformation specification. In a transformation step, the source application model is transformation-independent (with respect to a transformation specification from that level of models), since it relies on an abstract platform, which is itself transformation-independent (see Table 6). The target application model and the transformation arguments can be classified as transformation-specific. This can serve as a guideline to determine whether design decisions should be captured at the source application model level or at either transformation arguments or the target application model level.

## 6 Main conclusions and directives

From the analysis of the relations between the various models, we can conclude that:

- *Feedback dependencies between execution and preparation phases can be avoided by addressing generality requirements at the preparation phase*. Failure to address these requirements results in cycles between the execution and preparation phases;
- *Platform-independent and platform-specific models are interrelated, their dependencies defined by transformation*. The interrelation between PIMs and PSMs is addressed through iteration in the execution phase. An iteration in the execution phase allows a designer to gain insight into the implications of certain design decisions at the PIM-level.

Our analysis leads to the following directives for the design process:

- *Changes in PIM, PSM or transformation arguments must be accommodated in PIM, PSM or transformation arguments, but not in the abstract platform, concrete platform nor transformation specification.*
- *Dependencies between PIM and PSM are handled by iterations in the execution phase, leading to a stable application PIM that does not depend on platform-specific design decisions.*
- *Interdependent design decisions must be captured at the same level of platform-independence. Since some design decisions are platform-specific, this imposes constraints on the organization of models at different levels of platform-independence. We have illustrated the consequences of interdependent design decisions with an example in [1].*
- *The classification of models according to the various dimensions of concerns serves as a guideline to determine in which models design decisions should be captured.*

## References

1. Almeida, J.P.A., Dijkman, R., van Sinderen, M., Ferreira Pires, L.: On the Notion of Abstract Platform in MDA Development. In: Proceedings Eighth IEEE International Conference on Enterprise Distributed Object Computing (EDOC 2004). IEEE CS Press (2004) 253–263
2. Almeida, J.P.A., van Sinderen, M., Ferreira Pires, L., Quartel, D.: A systematic approach to platform-independent design based on the service concept. In: Proceedings Seventh IEEE International Conference on Enterprise Distributed Object Computing (EDOC 2003). IEEE CS Press (2003) 112–134
3. Baldwin, C.Y, Clark, K.B.: Design Rules, Volume 1, The Power of Modularity. MIT Press, Cambridge, MA (2000)
4. Baldwin, C.Y, Clark, K.B.: Modularity in the Design of Complex Engineering Systems, Harvard Business School Working Paper Series, No. 04-055 (2004)
5. Gardner, T., Griffin, C., Koehler, J., Hauser, R.: A review of OMG MOF 2.0: Query / Views / Transformations Submissions and Recommendations towards the final Standard, ad/03-08-02, OMG (2002)
6. Gavras, A., Belaunde, M., Ferreira Pires, L., Almeida, J.P.A.: Towards an MDA-based development methodology for distributed applications. In: Proceedings of the 1st European

Workshop on Model-Driven Architecture with Emphasis on Industrial Applications (MDA-IA 2004), CTIT Technical Report TR-CTIT-04-12, University of Twente, Enschede, The Netherlands (2004) 43–51
7. Object Management Group: MDA-Guide, V1.0.1, omg/03-06-01, OMG (2003)
8. Object Management Group: UML 2.0 Superstructure, ptc/03-08-02, OMG (2003)
9. Steward, D.V.: The Design Structure System: A Method for Managing the Design of Complex Systems. In: IEEE Transactions on Engineering Management, Vol. 28 (1981) 71–74
10. Warfield, J.N.: Binary Matrices in System Modeling. In: IEEE Transactions on Systems, Man, and Cybernetics, Vol. 3 (1973) 441–449
11. Yassine, A., Braha, D.: Complex Concurrent Engineering and the Design Structure Matrix Method. In: Concurrent Engineering, Vol. 11, No. 3, SAGE Publications (2003) 165–176

# From Mapping Specification to Model Transformation in MDA: Conceptualization and Prototyping

Slimane Hammoudi, Denivaldo Lopes

ESEO, Ecole Supérieure d'Electronique de l'Ouest
4, Rue Merlet de la Boulaye
BP 926, 49009 ANGERS cedex 01
{shammoudi, dlopes}@eseo.fr

**Abstract.** In this paper, we present in the first part our proposition for a clarification of the concepts of mapping and transformation in the context of Model Driven Architecture (MDA), and our approach for mapping specification and generation of transformation definition. In the second part, we present the application of our approach from UML to C#. We propose a metamodel for mapping specification and its implementation as a plug-in for Eclipse. Once mappings are specified between two metamodels (e.g. UML and C#), transformation definitions are generated automatically using transformation languages such as Atlas Transformation Language (ATL). We have applied this tool to edit mappings between UML and C# metamodels. Afterwards, we aim to use these mappings to generate ATL code to achieve transformations from UML into C#.

## 1 Introduction

The Object Management Group (OMG) has stimulated and promoted the adoption of the Model Driven Architecture (MDA™) [1] to define an approach to software development based on modeling and automated mapping of models to implementation. In this approach, models become the hub of development, separating platform independent characteristics (i.e. Platform-Independent Model - PIM) from platform dependent characteristics (i.e. Platform-Specific Model - PSM).
The MDA approach promises a number of benefits including business logic is protected against the changes in technologies, systems can evolve for meeting new requirements, old, current and new technologies can be harmonized, legacy systems could be integrated and harmonized with new systems.
In this approach, models are applied in all steps of development up to a target platform, providing source code, files of deployment and configuration, and so on. MDA proposes architecture to address the complexity of software development and maintenance, which has no precedents. It claims that software developers can create and maintain software artifacts with little effort. However, before this becomes a mainstream reality some issues in MDA approach need solutions such as mapping, transformation, handling of semantic distance between metamodels [3], bidirectional mapping [4], and so on.

In this paper, we use the term mapping as a synonym for correspondence between the elements of two metamodels, while transformation is the activity of transforming a source model into a target model in conformity with the transformation definition. In our approach, a transformation definition is generated from a mapping specification. The distinction between mapping specification and transformation definition is detailed in later sections. The objective of this paper is threefold. First, to provide a precise definition of the concepts of mapping and transformation. Second, to provide a general metamodel for mapping specification between two metamodels (source and target) in the context of MDA. Third, to present a tool based on eclipse enabling the edition of mappings and the generation of transformation definition from mapping specifications. We will apply this tool to C# Platform from UML as PIM.

This paper is organized in the following way. Section 2 is an overview of MDA and the main concepts behind this framework. Section 3 presents our proposition for the clarification of the concepts of mapping and transformation and our metamodel for mapping specification between two metamodels in the context of MDA. Section 4 introduce briefly a formalism for mapping and shows the implementation of our proposed metamodel through a plug-in for eclipse, and its application to C# platform from UML PIM. Section 5 concludes this paper and presents the future directions of our research.

## 2 Background

The OMG's Model Driven Architecture (MDA) is a new approach to develop large software systems in which the initial efforts aim to cover their functionalities and their behavior. MDA separates the modeling task of the implementation details, without losing the integration of the model and the development in a target platform. The key technologies of MDA are Unified Modeling Language (UML), Meta-Object Facility (MOF), XML Meta-Data Interchange (XMI) and Common Warehouse Metamodel (CWM) [1]. Together, they unify and simplify the modeling, the design, the implementation, and the integration of systems. One of the main ideas of MDA is that each model is based on a specific meta-model. Each meta-model precisely defines a domain specific language. Finally, all meta-models are based on a meta-metamodel. In the MDA technological space, this is the Meta-Object Facility (MOF). There are also standard projections on other technological spaces like XMI for projection on XML and Java Metadata Interface (JMI) for projection on Java. MDA also introduces other important concepts: Platform Independent Model (PIM), Platform-Specific Model (PSM), transformation language, transformation rules and transformation engine. These elements of MDA are depicted in figure 1.

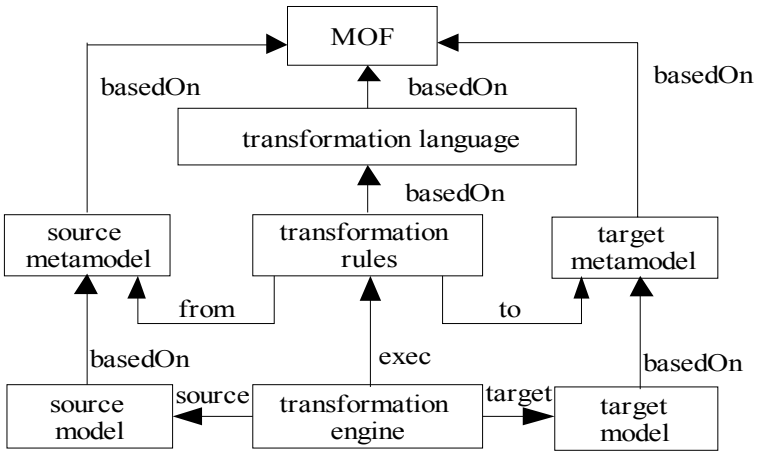



**Fig. 1.** Transformation in MDA

Each element presented in Figure 1 plays an important role in MDA. In our approach, MOF is the well-established meta-meta-model used to build meta-models. The PIM reflects the functionalities, the structure and the behavior of a system. The PSM is more implementation-oriented and corresponds to a first binding phase of a given PIM to a given execution platform. The PSM is not the final implementation, but has enough information to generate interface files, a programming language code, an interface definition language, configuration files and other details of implementation. Mapping from PIM to PSM determines the equivalent elements between two meta-models. Two or more elements of different meta-models are equivalents if they are compatible and they cannot contradict each other. Model transformation is realized by a transformation engine that executes transformation rules. Transformation rules specify how to generate a target model (i.e. PSM) from a source model (i.e. PIM).

To transform a given model into another model, the transformation rules map the source into the target meta-model. The transformation engine takes the source model, executes the transformation rules, and gives the target model as output. Using one unique formalism (e.g. MOF) to express all meta models is very important because this allows the expression of all sorts of correspondence between models based on separate meta-models. Transformations are one important example of such correspondence, but there are also others like traceability, etc. In other words, given m1(s)/Ma and m2(s)/Mb, where m1 is a model of a system s created using the meta-model Ma, and m2 is a model of the same system s created using the meta-model Mb, then a transformation can be de- fined as m1(s)/Ma → m2(s)/Mb. When Ma and Mb are based on the same meta-meta-model, the transformation may be expressed in a unique transformation language (i.e. a language independent of the meta-model). Furthermore, the transformation language itself may be considered as a domain-specific language. This has many interesting consequences. One of these is that a transformation program corresponds to an MDA model. We may thus easily consider higher-order transformations, i.e. transformations having other transformations as input and/or producing other transformations as output. One of the most popular meta-models is UML, but there are plenty of other meta-models being defined. For

example, there could be a meta-model of the Java language. Based on these two meta-models, it is possible to express a transformation from UML 1.5 class diagrams to Java 1.4.2 code. In fact, models have been used for a long time, but they remained disconnected from the implementation process.
The automatic generation of code to a specific language from a UML class diagram is not new either; some CASE tools give this support such as Poseidon for UML (http://www.gentleware.com). However developers still have to write all the application codes by hand. Moreover, when the application has evolved to acquire new capabilities or adapt to new technologies, these tools cannot help the developer, and the model is used only as documentation. An Integrated Development Environment (IDE) provides a set of tools integrated on a single user interface that often comprises a sophisticated text editor, a graphical editor for GUI, an editor to database tables, a compiler and a debugger, e.g. IBM's WebSphere Studio and Microsoft's Visual Studio .NET. An IDE can aid the software development, but only in the programming phase (i.e. it is based on code-centric approach). A tool powered with MDA will be enabled to support the system development throughout its life cycle. The development of large software systems can take some suggested benefits (some benefits are still not proven) from MDA:

- The same PIM can be used many times to generate models on different platforms (PSMs) [8];
- Many views of the same system, e.g. many abstraction levels or details of implementation. We de-fine abstraction levels as the possibility to see a system (e.g. applications and business process) fragmented in many different and interlinked levels, each level detaching important characteristics of the same system;
- Enhancement of the portability and of the interoperability of systems in the level of models;
- Preservation of the business's logic against the changes or evolution of technology ;
- An uniform manner for business models and for technologies to evolve together;
- Prevention against error-prone factors linked to manual design of systems [7];
- Increase the return on technology investments;
- Enhancement of the reengineering, i.e. it assists the recuperation of business's logic from source codes or from implementation environments;
- Enhancement of the interaction and of the migration between different technological spaces.

Apart from these benefits, the approach using models forces the architects to think about the architecture and the model behind the system in development, whereas a code centric approach makes architects concentrated on the code, so they consequently forget the main properties of the system. Other case studies have shown some benefits of the MDA Approach. In [11], the authors have demonstrated that the development of a case study (i.e. J2EE PetStore) using a MDA tool is 35% faster than the development using a code centric approach (i.e. using a non powered-MDA tool).

## 3 Mapping and Transformation

Nowadays, MDA suffers from a lack of agreement on terminology, especially concerning the concepts of mapping and transformation. In MOF QVT [6], mapping is defined as *specification of a mechanism for transforming the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel*. In MDA distilled book [13], mapping is defined as *the application or execution of a mapping function in order to transform one model to another*, and mapping function is defined as *a collection of mapping rules that defines how a particular mapping works*. In both references and others discussed in [7], the concept of mapping and transformation are not so clear, since these terms can refer to many different concepts. Moreover, they are usually defined without explicit distinction between them. The table 1 presented in [7], and extended briefly here, illustrates in an obvious manner that the terminology related with the transformation and mapping concepts is really immature.

**Table 1:** Equivalencies between terms according to the Transformation Pattern

| | **Transf. Instance** | **Tranf. Function** | **Transf. Model** | **Transf. Program** | **Transf. Progr. Lang** | **Transf. Interp** |
|---|---|---|---|---|---|---|
| **MDA Guide [2]** | *Transfo* | *Mapping Instance* | *Mapping* | | *Mapping Language* | |
| **MDA Distilled [13]** | *Mapping* | *Mapping rule* | *Mapping function* | | | |
| **MDA Explained[10]** | *Transfo. Mapping* | *Mapping/Transf rule* | *Transfo. Definition* | | | *Transfo. tool* |
| **QVT DSTC [6]** | *Tracking* | *Transfo. rule* | *Transfo.* | | | *Transfo. Engine* |
| **QVT Partners [14]** | | | *Transfo. Relation* | *Mapping Relation* | | |
| **[17]** | | *Mapping* | *Model of Mapping* | | *Mapping Formalism* | |
| **[18]** | *Transfo.* | | *Transfo. Pattern* | | | |
| **[19]** | | | | *Transfo. Spec* | | *Transformer* |
| **[20]** | *Transfo. Process* | | *Transfo. Descr* | | | |

According to our vision, the concepts of mapping and transformation should be explicitly distinguished, and together could be involved in the same process that we denominate transformation process. In fact, in the transformation process, the mapping specification precedes the transformation definition. A mapping specification is a definition (the most declarative as possible) of the correspondences between metamodels (i.e. a metamodel for building a PIM and another for building a PSM). Transformation definition contains a minute description to transform a model

into another using a hypothetic or concrete transformation language. Hence, in our approach the transformation process of a PIM into a PSM can be structured in two stages: mapping specification and transformation definition. Finally, we define the term transformation as the manual or automatic generation of a target model from a source model, according to a transformation definition. From a conceptual point of view, the explicit distinction between mapping specification and transformation definition remains in agreement with the MDA philosophy, i.e. the separation of concerns. Moreover, a mapping specification could be associated with different transformation definitions, where each transformation definition is based on a giving transformation definition metamodel. Figure 2 illustrates the different concepts of MDA according to our vision where mapping specification is a mapping model, and transformation definition is a transformation model. In this figure, a mapping model is based on its metamodel, and it relates two metamodels (left and right). A transformation model is based on its transformation metamodel, and it is generated from a mapping model. A transformation engine takes a source model as input, and it executes the transformation program to transform this source model into the target model.

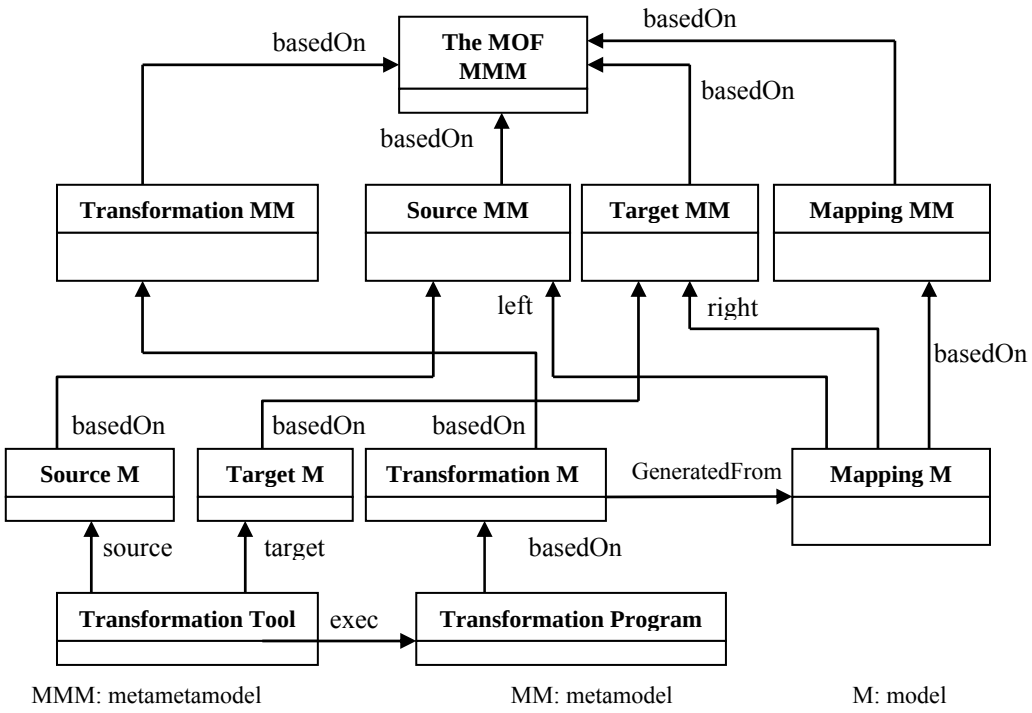



**Fig. 2:** Transformation Process within MDA: from Mapping to Transformation

Several research projects have studied the mapping specification between metamodels [9] [12] [18]. However, the ideas around mapping specification are not sufficiently developed to create efficient tools to enable automatic mappings.

Nowadays, transformation languages are not yet very well explored to make choices about a standard transformation language such as desired by OMG [2]. In the next few years, the submitted propositions [6] [14] in response to QVT RFP might converge to a standard language, which will provide a new step forward in the evolution of MDA. However, wisdom tells us that one problem can be resolved using different solutions, but one solution for all problems does not exist. Thus, it is clear that this standard language will not provide a sufficient solution for all types of model transformations around MDA. However, this will not be a limitation for applying MDA, because a transformation language is also a model, thus one transformation language can also be transformed into another transformation language. A priori, transformations between transformation languages seem unnecessary and unproductive. However, several examples such as Structured Query Language (SQL) (i.e. a standard query language for manipulating databases) have demonstrated that a standard is beneficial, because it establishes a unique and well-known formalism for understanding a problem and its solution. On the one hand, SQL provides a universal language for manipulating databases. On the other hand, SQL can be transformed into a proprietary language for execution into a database engine. A transformation from SQL into a proprietary language provides some benefits such as improved performance, reduction of memory-use, and so on. Making an analogy between SQL and a standard transformation language, we can expect that a standard transformation language can provide some benefits without imposing severe limitations. Mapping and transformation have been studied for a long time ago in the database domain. However, they have taken another dimension with the sprouting of MDA. This not means that they are well studied and done to be applied in the MDA context. In fact, mapping specification and transformation definition are not yet an easy task. Moreover, tools to enable the automatic creation of mapping specification and automatic generation of transformation definition are still under development.
In the next section, we start briefly presenting a foundation for mapping and afterwards we discuss our proposition for specifying mappings (i.e. correspondences between metamodels). This approach for mapping is based on a metamodel and implemented as a tool on Eclipse. This tool provides mapping support that is a preliminary step before the generation of a transformation definition.

## 4 Foundations and Prototyping

In this section, we present our proposition for specifying mappings (i.e. correspondences between metamodels). This approach for mapping is based on a metamodel and implemented as a tool on Eclipse. This tool provides support for mapping, which is a preliminary step before the creation of a transformation definition, using ATL. We have applied this tool for the different cases presented previously.
The creation of mapping specification and transformation definition is not an easy task. In addition, the manual creation of mapping specification and transformation definition is a labor-intensive and error-prone process [12]. Thus the creation of an automatic process and tools for enabling them is an important issue. Some

propositions enabling the mapping specification have been based on heuristics [12] (for identifying structural and naming similarities between models) and on machine learning (for learning mappings). Other propositions enabling transformation definition have been based on graph theory . Mapping specification is not a new issue in computer science.

For a long time, the database domain has applied mappings between models (i.e. schema) and transformation from different conceptual models, e.g. entity-relationship (ER), into logical or physical models (relational-tables and SQL schema). However, these same issues have taken a new dimension with the sprouting of MDA, because models become the basis to generate software artifacts (including code) and in order to transform one model into another model, mapping specification is required. So, both mapping specification and transformation definition have been recognized as important issues in MDA.

### 4.1 A metamodel for mapping

In order to define a mapping, we need a metamodel, which enables:

- Identification of what elements has similar structures and semantics to be mapped.
- Explanation of the evolution in time of the choices taken for mapping one element into another element.
- Bi-directional mapping. It is desirable, but is often complex [10].
- Independence of model transformation language.
- Navigation between the mapped elements.

Figure 3 presents our proposition of a metamodel for mapping specification. A complete definition of this metamodel is presented in [5].

In this metamodel, we consider that a mapping can be unidirectional or bi-directional. In unidirectional mapping, a metamodel is mapped into another metamodel. In bi-directional mapping, the mapping is specified in both directions. Thus, as presented previously, we prefer refer to the two metamodels in a mapping as left or right metamodels.

A central element in this metamodel is the element `Correspondence.` This element is used to specify the correspondence between two or more elements, i.e. left and right element. The correspondence has a filter that is an OCL expression. When bidirectional is false, a mapping is unidirectional (i.e. left to right), and when it is false it is bidirectional (i.e. in both directions). It has two `TypeConverters` identified by `typeconverterRL` and `typeconverterLR`. `typeconverterRL` enables the conversion of the elements from a right metamodel into the elements from a left metamodel. `typeconverterLR` enables the conversion of the elements from a left metamodel into the elements from a right metamodel. We need often specify only `the typeconverterLR`.

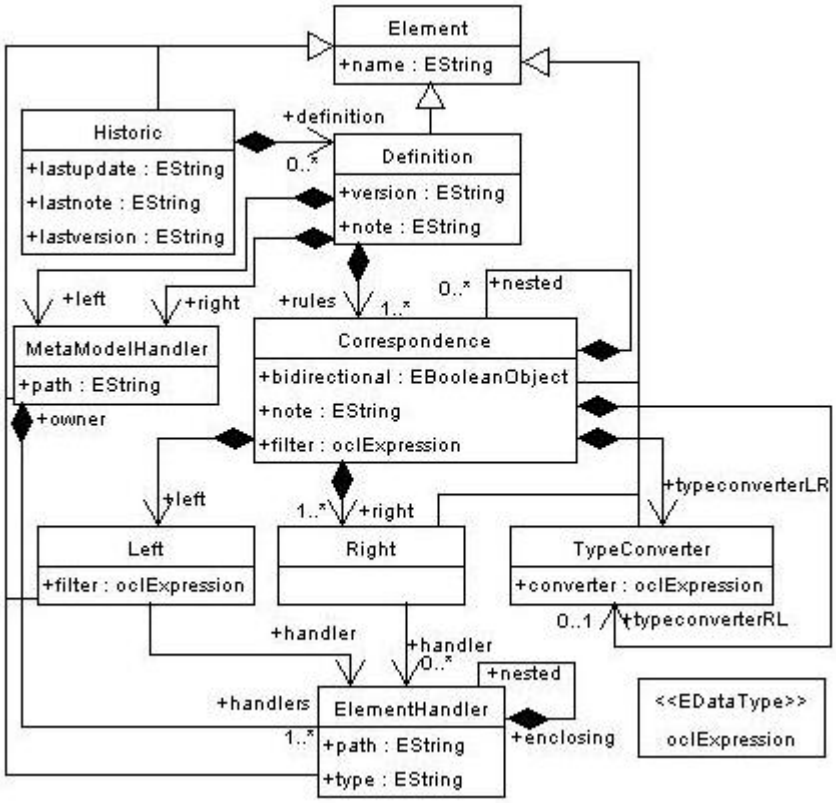



**Fig. 3.** Metamodel for Mapping Specification

### 4.2 A Plug-in for Eclipse

Eclipse Modeling Framework (EMF) is a modeling framework and code generation facility for supporting the creation of tools and applications driven by models [15]. EMF represents the efforts of Eclipse Tools Project to take into account the driven model approach. In fact, MDA and EMF are similar approaches for developing software systems, but each one has different technologies. MDA was first designed by OMG using MOF and UML, while EMF is based on Ecore and stimulates the creation of specific metamodels.

A tool supporting our proposed metamodel for mapping should provide:

- Simplification for visualizing mappings. In order to specify a mapping, two metamodels are necessary. From experience, metamodels have generally a considerable number of elements and associations. So the visualization becomes complex, putting two metamodels so large side by side and the mapping in the center. A tool should allow the creation of views, navigation and encapsulation

of details unnecessary for each mapping in order to facilitate the visualization and comprehension of the mapping without modifying the involved metamodels.

- Creation of transformation definition from mapping specification. A mapping specification is a model itself, and then it can also be transformed into another model. For example, a mapping model can be transformed into a transformation model.

Our proposed tool supports all these characteristics, except the *semi-automatic matching* which is the next step for its improvement.

Figure 4 shows our plug-in for Eclipse. This tool is denominated mapping Modeling Tool (MMT). The tool presents a first metamodel on the left side, a mapping model in the center, and a second metamodel on the right. In this figure, the UML metamodel (fragment) is mapped into a C# metamodel (fragment). At the bottom, the property editor of mapping model is shown. A developer can use this property editor to set the properties of a mapping model. Before specifying mapping using our tool, we need create metamodels based on Ecore. Some tools support the editing of a metamodel based on Ecore such as Omondo [15] or the eCore editor provided with EMF. The application of our tool using UML and C# metamodel can be explained in the following steps:

1. We created a project in eclipse and we imported the UML and C# metamodel into this project.
2. We used a wizard to create a mapping model. In this step, we chose the name for the mapping model, the encoding of the mapping file (e.g. Unicode and UTF- 8), the metamodels files in the format XMI.
3. The UML and C# metamodels are loaded from the XMI files, and the mapping model is initially created, containing the elements Historic, Definition, and left and right MetamodelHandlers. For each MetamodelHandler is also created ElementHandlers that are references to the elements of the corresponding metamodel.
4. We edit the mapping model. First, we define the inter-relationships between the metamodels creating correspondences between their elements. Second, we create for each correspondence nested correspondence. Third, for each nested correspondence, we create one Left and one or more Right elements. In addition, each Left and Right element has a ElementHandler. If it is necessary, the TypeConverter is created to explicit the casting between two mapped elements.
5. The mapping model can be validated according to its metamodel, and it can be used to generate a transformation definition (e.g. using ATL language).

This tool can export a mapping model as transformation definition. For the moment, we have implemented a generator for ATL [8], but we envisage creating generators to other model transformation languages such as YATL [4], in order to evaluate the power of our proposed metamodel for mapping.

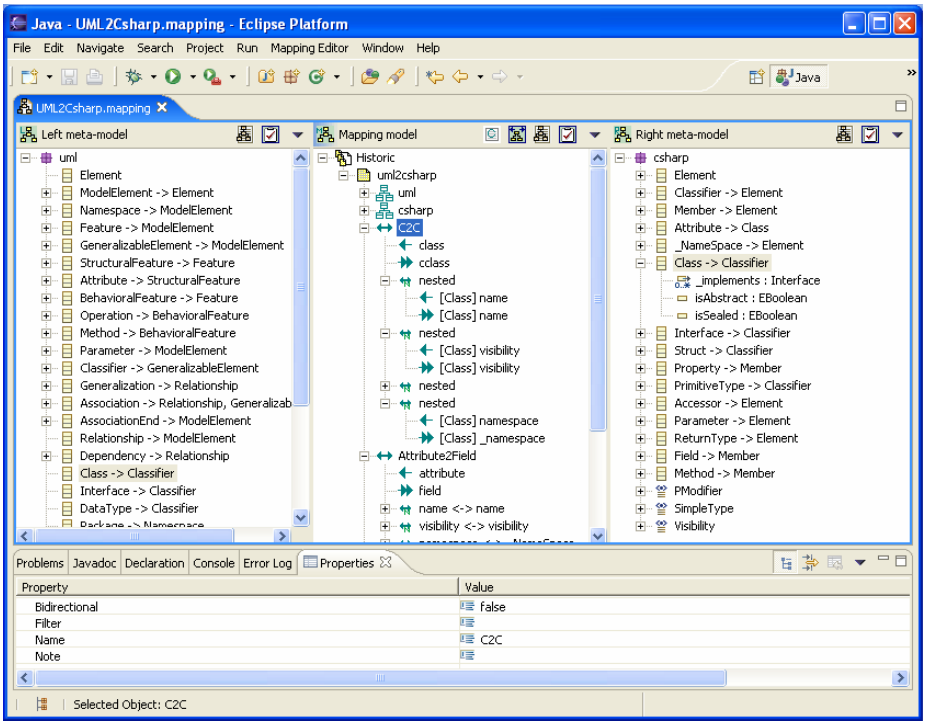



**Fig. 4.** Mapping Modelling Tool (MMT): From UML to C# metamodel

## 5 Conclusion

In this paper, we have discussed the MDA approach providing a detailed description of transformation process, distinguishing mapping and transformation. If transformation is the heart and soul of MDA [16], and transforming a PIM into a PSM requires finding correspondences between metamodels, then mapping specification is also another important issue within MDA context. We have proposed a metamodel for mapping and a tool (i.e. MMT) to support mappings. To illustrate our tool, we have specified mappings between UML as PIM and C# as PSM. According to model management algebra, a mapping is generated using an operator called match, which takes two metamodels as input and returns a mapping between them. The schema matching was not yet integrated in our plugin, because, at this stage, we are more interested in addressing the creation of mappings driven by models.

## References


1. OMG: Model Driven Architecture (MDA)- document number ormsc/2001-07-01. (2001)
2. OMG, « MDA Guide Version 1.0.1 », OMG/2003-06-01, June 2003.
3. Kent, S., Smith, R.: The Bidirectional Mapping Problem. Electronic Notes in Theoretical Computer Sciences 82 (2003)
4. Patrascoiu, O.: Mapping EDOC to Web Services using YATL. 8th IEEE Enterprise Distributed Object Computing Conference (EDOC 2004) (2004)
5. Lopes, D., Hammoudi, S. Bézivin, J, Jouault, F.: Mapping Specification in MDA: From Theory to Practice. INTEROP-ESA'2005 Conference (February 23-25, 2005)
6. DSTC, IBM, and CBOP. *MOF Query / Views / Transformations Second Revised Submission*, January 2004. ad/2004-01-06.
7. J. M. Favre. Towards a Basic Theory to Model Driven Engineering. *UML 2004 - Workshop in Software Model Engineering (WISME 2004)*, 2004.
8. Bézivin, J., Dupre, G., Jouault, F., Pitette, G., Rougui, J.E.: First Experiments with the ATL Model Transformation Language: Transforming XSLT into XQuery. 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture (2003)
9. Jan Hendrick Hausmann, S.K.: Visualizing Model Mappings in UML. ACM 2003 Symposium on Software Visualization (SOFTVIS 03) (2003) 169–178
10. A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley, August 2003.
11. Middleware Company: Model Driven Development for J2EE Utilizing a (MDA) Approach.www.middleware-company.com/casestudy.
12. Madhavan, J., Bernstein, P.A., Domingos, P., Halevy, A.Y.: Representing and Reasoning about Mappings between Domain Models. Eighteenth National Conference on Artificial intelligence (AAAI'02) (2002) 80–86
13. S. J. Mellor, K. Scott, A. Uhl, and D. Weise. *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley, 1st edition, March 2004.
14. QVT-Merge Group. *Revised submission for MOF 2.0 Query/Views/Transformations RFP (ad/2002-04-10)*, April 2004. Available at http://www.omg.org/docs/ad/04-04- 01.pdf.
15. Omondo. *Omondo Eclipse UML*, October 2004. Available at http://www.omondo.com.
16. Sendall S, Kozaczynski W : Model Transformation – the Heart and Soul of Model Driven Software Development.
17. G. Caplat and J. L. Sourrouille. Model Mapping in MDA. *Workshop in Software Model Engineering (WISME2002)*, 2002.
18. S.R.Judson, R.B.France, D.L.Carver. "Specifying Model Transformation at the Metamodel Level", WISME 2003.
19. I Kurtev, K.Van den Berg. A synthesis based approach to Transformation in an MDA Software Development Process.
20. M. Peltier. Techniques de Transformation de Modèles Basées sur la méta-modélisation. Thèse de Doctorat, Université de Nantes, 2003.

# A Formal Semantics for the Business Process Execution Language for Web Services

Roozbeh Farahbod, Uwe Glässer and Mona Vajihollahi

Software Technology Lab
School of Computing Science
Simon Fraser University
Burnaby, B.C., Canada
{rfarahbo/glaesser/mvajihol}@cs.sfu.ca

**Abstract.** We define an abstract operational semantics for the Business Process Execution Language for Web Services (BPEL) based on the *abstract state machine* (ASM) formalism. This way, we model the dynamic properties of the key language constructs through the construction of a *BPEL abstract machine* in terms of a distributed real-time ASM. Specifically, we focus here on the *process execution model* and the underlying *execution lifecycle* of BPEL activities. The goal of our work is to provide a well defined semantic foundation for establishing the key language attributes. The resulting abstract machine model provides a comprehensive and robust formalization at three different levels of abstraction.

**Keywords:** Web Services Orchestration, BPEL4WS, Abstract Operational Semantics, Abstract State Machines, Requirements Specification

## 1 Introduction

In this paper, we present an abstract operational semantics of the XML based Business Process Execution Language for Web Services (BPEL4WS) [1], a novel Web Services orchestration language proposed by OASIS [2] as a future standard for the e-business world. BPEL4WS, or BPEL for short, provides distinctive expressive means for describing the process interfaces of Web based business protocols and builds on existing standards and technologies for Web Services. It is defined on top of the service interaction model of W3C's Web Services Description Language (WSDL) [3]. A BPEL business process orchestrates the interaction between a collection of abstract WSDL services exchanging messages over a communication network.

Based on the *abstract state machine* (ASM) formalism [4], we define a *BPEL abstract machine*, called $\mathsf{BPEL}_{\mathcal{AM}}$, as a concise and robust semantic framework for modeling the key language attributes in a precise and well defined form. That is, we formalize dynamic properties of the Web Services interaction model of a BPEL business process in terms of finite or infinite abstract machine *runs*. Due to the concurrent and reactive nature of Web Services and the need for dealing with time related aspects in coordinating distributed activities, we combine an asynchronous execution model with an abstract notion of real time. The resulting computational model is referred to as a *distributed real-time ASM*. Our model captures the dynamic properties of the key

language constructs defined in the language reference manual [1], henceforth called the LRM, including concurrent control structures, dynamic creation and termination of service instances, communication primitives, message correlation, event handling, and fault and compensation handling.

The goal of our work is twofold. First and foremost, BPEL$_{\mathcal{AM}}$ provides a firm semantic foundation for checking the consistency and validity of the language definition by conceptual means and by analytical means. Formalization is crucial for identifying and eliminating deficiencies that otherwise remain hidden in the informal language definition of the LRM [2, Issue #42]: "*There is a need for formalism. It will allow us to not only reason about the current specification and related issues, but also uncover issues that would otherwise go unnoticed. Empirical deduction is not sufficient.*"

Second, we address pragmatic issues resulting from previous experience with other industrial standards, including the ITU-T language SDL[1] [6] and the IEEE language VHDL [7]. An important observation is that formalization techniques and supporting tools for practical purposes such as standardization call for a gradual formalization of abstract requirements with a degree of detail and precision as needed [8]. To avoid a gap between the informal language definition and the formal semantics, the ability to model the language definition *as is* without making compromises is crucial. Consequently, we adopt here the view and terminology of the LRM, effectively formalizing the intuitive understanding of BPEL as directly as possible in an objectively verifiable form.

The result of our work is what is called an *ASM ground model* [4] of BPEL. Intuitively, ground models serve as 'blueprints' for establishing functional software requirements, including their elicitation, clarification and documentation. Constructing such a ground model requires a major effort — especially, as a clear architectural view, which is central for dealing with complex semantic issues, is widely missing in the BPEL language definition.

The paper is organized as follows. Section 2 briefly summarizes the formal semantic framework. Section 3 introduces the core of our hierarchically defined BPEL$_{\mathcal{AM}}$, and Section 4 then addresses important extensions to the BPEL$_{\mathcal{AM}}$ core. Section 5 discusses related work, and Section 6 concludes the paper.

## 2 Distributed Real-time ASM

We briefly outline the formal semantic framework at an intuitive level of understanding using common notions and structures from discrete mathematics and computing science. For details, we refer to the existing literature on the theory of abstract state machines [9] and their applications [4].[2]

We focus here on the asynchronous ASM model, called distributed abstract state machine (DASM), as formal basis for modeling concurrent and reactive system behavior in terms of abstract machine *runs*. A DASM $M$ is defined over a given vocabulary $V$ by its program $P_M$ and a non-empty set $I_M$ of initial states. $V$ consists of symbols denoting the various semantic objects and their relations in the formal representation

[1] Our ASM semantic model of SDL is part of the current SDL standard defined by the International Telecommunication Union [5].

[2] See also the ASM Web site at www.eecs.umich.edu/gasm.

of $M$, where we distinguish *domain symbols*, *function symbols* and *predicate symbols*. Symbols that have a fixed interpretation regardless of the state of $M$ are called *static*; those that may have different interpretations in different states of $M$ are called *dynamic*. A state $S$ of $M$ yields a valid interpretation of all the symbols in $V$.

Concurrent control threads in an execution of $P_M$ are modeled by a dynamic set AGENT of autonomously operating *agents*. Agents of $M$ interact with each other by reading and writing shared locations of global machine states, where the underlying semantic model regulates such interactions so that potential conflicts are resolved according to the definition of *partially ordered runs* [4].

$P_M$ consists of a statically defined collection of agent programs, each of which defines the behavior of a certain *type* of agent in terms of state transition rules. The canonical rule consists of a basic update instruction of the form $f(t_1, t_2, ..., t_n) := t_0$, where $f$ is an n-ary dynamic function symbol and the $t_i s$ ($0 \leq i \leq n$) are terms. Intuitively, one can conceive a dynamic function as a *function table* where each row associates a sequence of argument values with a function value. An update instruction specifies a pointwise function update, i.e., an operation that replaces an existing function value by a new value to be associated with the given arguments.

Finally, $M$ models the embedding of a system into a given environment — the *external world* — through actions and events as observable at interfaces. The external world affects operations of $M$ through externally controlled or *monitored* functions. Such functions change their values dynamically over runs of $M$, although they cannot be updated by agents of $M$. A typical example is the representation of time by means of a nullary monitored function $now$ taking values in a linearly ordered domain TIME. Intuitively, $now$ yields the time as measured by some external clock.

## 3 BPEL Abstract Machine

This section introduces the core components of $\mathsf{BPEL}_{\mathcal{AM}}$ architecture and the underlying abstraction principles starting with a brief characterization of the key language features as defined in [1]. We then present BPEL's process execution model and its decomposition into *execution lifecycles* of basic and structured activities. As a concrete example of a structured activity dealing with concurrency and real-time aspects, we consider the *pick* activity. The architectural view, the decomposition into execution lifecycles, and the model of *pick* are new and not contained in [10].

BPEL introduces a stateful model of Web Services interacting by exchanging sequences of messages between business partners. A BPEL process and its partners are defined as abstract WSDL services using abstract messages as defined by the WSDL model for message interaction. The major parts of a BPEL process definition consist of (1) *partners* of the business process (Web services that this process interacts with), (2) a set of *variables* that keep the state of the process, and (3) an *activity* defining the logic behind the interactions between the process and its partners. Activities that can be performed by a business process are categorized into *basic* activities, *structured* activities and *scope-related* activities. Basic activities perform simple operations like *receive*, *reply*, *invoke* and others. Structured activities impose an execution order on a collection

of activities and can be nested. Scope-related activities enable defining logical units of work and delineating the reversible behaviour of each unit.

**Dynamic Process Creation** A BPEL process definition works as a template for creating business process instances. Process creation is implicit and is done by defining a *start activity*, which is either a *receive* or a *pick* activity that is annotated with *'createInstance = yes'*, causing a new process instance to be created upon receiving a matching message. That is, when a new instance of a business process is created, it starts its execution by receiving the message that triggered its creation.

**Correlation and Data Handling** A Web service consists of a number of business process instances; thus, the messages arriving at a specific port must be delivered to the correct process instance. BPEL introduces a generic mechanism for dynamic binding of messages to process instances, called *correlation*.

**Long Running Business Transactions** Business processes normally perform transactions with non-negligible duration involving local updates at business partners. When an error occurs, it may be required to reverse the effects of some or even all of the previous activities. This is known as *compensation*. The ability to compensate the effects of previous activities in case of an exception enables so-called Long-Running (Business) Transactions (LRTs).

### 3.1 Abstract Machine Architecture

Logically, $\mathsf{BPEL}_{\mathcal{AM}}$ consists of three basic building blocks referred to as *core*, *data handling extension*, and *fault and compensation extension* (Figure 1). The *core* handles dynamic process creation/termination, communication primitives, message correlation, concurrent control structures, as well as the following activities: *receive*, *reply*, *invoke*, *wait*, *empty*, *sequence*, *switch*, *while*, *pick* and *flow*. The *core* does not consider data handling, fault handling, and compensation behavior. Rather these aspects are treated as extensions to the core (see Section 4). Together with the *core* these extensions form the complete $\mathsf{BPEL}_{\mathcal{AM}}$.

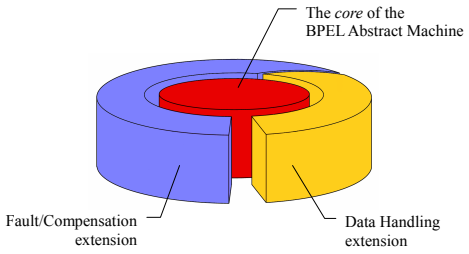



**Fig. 1.** $\mathsf{BPEL}_{\mathcal{AM}}$ Behavioural Decompositionam

The vertical organization of the machine architecture consists of three layers, called *abstract* model, *intermediate* model and *executable* model. The abstract model formally

sketches the behavior of the key BPEL constructs, while the intermediate model, obtained as the result of the first refinement step, provides a complete formalization. Finally, the executable model provides an abstract executable semantics implemented in AsmL [8]. A GUI facilitates experimental validation through simulation and animation of abstract machine runs.

Figure 2 shows an abstract view of the underlying Web Services interaction model. A BPEL document abstractly defines a Web service consisting of a collection of business process instances. Each such instance interacts with the external world through two interface components, called *inbox manager* and *outbox manager*. The inbox manager handles all the messages that arrive at the Web service. If a message matches a request from a local process instance waiting for that message, it is forwarded to this process instance. Additionally, the inbox manager also deals with new process instance creation. The outbox manager, on the other hand, forwards outbound messages from process instances to the network.

Inbox manager, outbox manager, and process instances are modeled by three different types of DASM agents: the *inbox manager agent*, the *outbox manager agent*, and one uniquely identified *process agent* for each of the process instances.

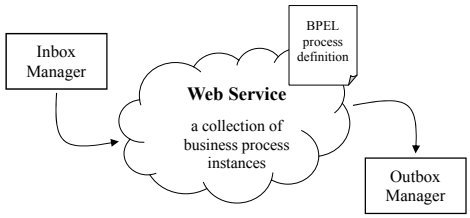



**Fig. 2.** High-level Structure of $\mathsf{BPEL}_{\mathcal{AM}}$

### 3.2 Activity Execution Lifecycle

Intuitively, the execution of a process instance is decomposed into a collection of execution lifecycles for the individual BPEL activities. We therefore introduce *activity agents*, created dynamically by process agents for executing structured activities. Each activity agent dynamically creates additional activity agents for executing nested, structured activities. Similarly, it creates auxiliary activity agents for dealing with concurrent control threads (like in *flow* and *pick*[3]). For instance, to concurrently execute a set of activities, a flow agent assigns each enclosed activity to a separate *flow thread agent* [10]. At any time during the execution of a process instance, the DASM agents running under control of this process agent form a tree structure where each of the sub-agents monitors the execution of its child agents (if any) and notifies its parent agent in case of

[3] One may argue that *pick* is not a concurrent control construct, but as we will see in Section 3.3, it can naturally be viewed as such.

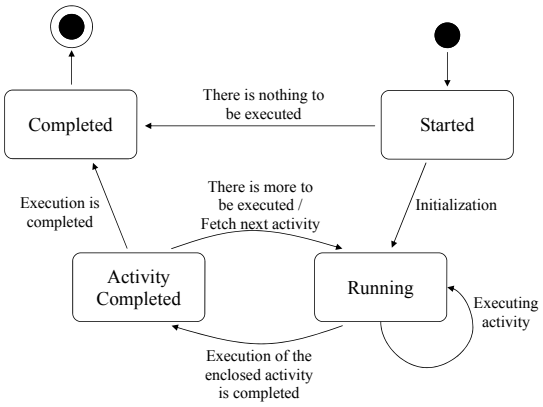



**Fig. 3.** Activity Execution Lifecycle: $\mathsf{BPEL}_{\mathcal{AM}}$ core

normal completion or fault. This structure provides a general framework for execution of BPEL activities. The DASM agents that model BPEL process execution are jointly called *kernel agents*. They include process agents and subprocess agents. In the *core*, however, subprocess agents are identical to activity agents.

Figure 3 illustrates the normal activity execution lifecycle of kernel agents in the $\mathsf{BPEL}_{\mathcal{AM}}$ core. When created, a kernel agent is in the *Started* mode. After initialization, the kernel agent starts executing its assigned task by switching its mode to *Running*. Upon completion, the agent switches its mode to *Activity-Completed* and decides (based on the nature of the assigned task) to either return to the *Running* mode or finalize the execution and become *Completed*. Activity agents that may execute more than one activity (like *sequence*) or execute one activity more than once (like *while*) can switch back and forth between the two modes *Activity-Completed* and *Running*.

### 3.3 Pick activity

A *pick* activity identifies a set of events and associates with each of these events a certain activity. Intuitively, it waits on one of the events to occur and then performs the respective activity; thereafter, the *pick* activity no longer accepts any other event. [4] There are basically two different types of events: *onMessage* events and *onAlarm* events. An onMessage event occurs as soon as a related message is received, whereas an onAlarm event is triggered by a timer mechanism waiting *'for'* a certain period of time or *'until'* a certain deadline is reached.

In $\mathsf{BPEL}_{\mathcal{AM}}$, each *pick* activity is modeled by a separate activity agent, called *pick agent*. A pick agent is assisted by two auxiliary agents, a *pick message agent* that is waiting for a message to arrive, and a *pick alarm agent* that is watching a timer. We formalize the semantics of the *pick* activity in several steps, each of which addresses

[4] Regarding the case that several events occur at a time, the LRM is somewhat loose declaring that the choice "is dependent on both timing and implementation." [1]

a particular property, and then compose the resulting DASM program, called PickProgram in which *self* refers to a pick agent executing the program.

Pick Agent

**PickProgram** ≡
  **case** *execMode*(*self*) **of**
    *Started* → PickAgentStarted
    *Running* → PickAgentRunning
    *ActivityCompleted* → FinalizePickAgent
    *Completed* → **stop** *self*

When created, the pick agent is in the *Started* mode and initializes its execution by creating a pick alarm agent and a pick message agent. It then switches its mode to *Running* and waits for an event to occur — either a message arrived or a timer expired.

Pick Agent

**PickAgentRunning** ≡
  **if** *normalExecution*(*self*) **then**
    **onsignal** $s$ : AGENT_COMPLETED
      *execMode*(*self*) := *ActivityCompleted*
    **otherwise**
      **if** *chosenAct*(*self*) = *undef* **then**
        **choose** *dsc* ∈ *occurredEvents*(*self*) **with** *MinTime*(*dsc*)
          *chosenAct*(*self*) := *onEventAct*(*edscEvent*(*dsc*))
          // *onEventAct* is the activity associated with an event
      **else**
        ExecuteActivity(*chosenAct*(*self*))

Depending on the event type, either the pick message agent or the pick alarm agent notifies the pick agent by adding an *event descriptor* to the *occuredEvents* set of the pick agent. An event descriptor contains information on the event such as the time of its occurrence. When an event occurs, the pick agent updates the function *chosenAct* (with initial value *undef*) with the activity associated with the event. Once the activity is chosen (*chosenAct(self)* $\neq$ *undef*), the pick agent performs the chosen activity and remains *Running* until the execution of the chosen activity is completed as indicated by a predicate *chosenActCompleted*. It then switches its execution mode to *Activity-Completed*.

Finalizing a running pick agent includes informing its parent agent that the execution is completed and changing the execution mode to *Completed*. As illustrated in Figure 3, the *Completed* mode leads to the agent's termination.

Due to the space limitations, we do not show here the definitions of PickAgent-Started, FinalizePickAgent, as well as the programs of the pick message and the pick alarm agents, but refer to [11, 12] for a complete description.

## 4 Extensions to the BPEL$_{\mathcal{AM}}$ Core

For a clear separation of concerns and also for robustness of the formal semantic model, the aspects of data handling, fault handling and compensation behavior are carefully

separated from the core of the language. To this end, the core of $\mathsf{BPEL}_{\mathcal{AM}}$ provides a basic, yet comprehensive, model for *abstract processes* in which data handling focuses on protocol relevant data in the form of correlations while payload data values are left unspecified [1].

Compensation and fault handling behavior is a fairly complex issue in the definition of BPEL. An in-depth analysis in fact shows that the semantics of fault and compensation handling, even when ignoring all the syntactical issues, is related to more than 40 individual requirements spread out all over the LRM. These requirements (some of them comprise up to 10 sub-items) address a variety of separate issues related to the core semantics, general constraints, and various special cases (see [2]). A thorough treatment of the extensions is beyond the space limitations of this paper. Thus, we present an overview of the fault handling behavior in the following sections and refer to [11] for a comprehensive description.

### 4.1 Scope activity

The *scope* activity is the core construct of data handling, fault handling, and compensation handling in BPEL. A *scope* activity is a wrapper around a logical unit of work (a block of BPEL code) that provides local variables, a fault handler, and a compensation handler. The fault handler of a scope is a set of *catch* clauses defining how the scope should respond to different types of faults. A compensation handler is a wrapper around a BPEL activity that compensates the effects of the execution of the scope. Each scope has a primary activity which defines the normal behavior of the scope. This activity can be any basic or structured activity. BPEL allows scopes to be nested arbitrarily. In $\mathsf{BPEL}_{\mathcal{AM}}$, we model scopes by defining a new type of activity agents, called *scope agents*.

Fault handling in BPEL can be thought of as a mode switch from the normal execution of the process [1]. When a fault occurs in the execution of an activity, the fault is thrown up to the innermost enclosing scope. If the scope handles the fault successfully, it sends an *exited* signal to its parent scope and ends gracefully, but if the fault is re-thrown from the fault handler, or a new fault has occurred during the fault handling procedure, the scope sends a *faulted* signal along with the thrown fault to its parent scope. The fault is thrown up from scopes to parent scopes until a scope handles it successfully. A successful fault handling switches the execution mode back to normal. If a fault reaches the global scope, the process execution terminates [1].

The normal execution lifecycle of the process execution model (Figure 3) needs to be extended to comprise the fault handling mode of BPEL processes. The occurrence of a fault causes the kernel agent (be it an activity agent or the main process) to leave its normal execution lifecycle and enter a fault handling lifecycle. Figure 4 illustrates the extended execution lifecycle of BPEL activities.

In $\mathsf{BPEL}_{\mathcal{AM}}$, whenever a sub-process agent encounters a fault, the agent leaves its normal execution mode and enters the *Execution-Fault* mode. If this agent is not a scope agent, it informs its parent agent of the fault and stays in the *Execution-Fault* mode until it receives a notification for termination. On the other hand, if the faulted agent is a scope agent, it terminates its enclosing activity, creates a fault handler, assigns the fault to that handler, and switches to the *Fault-Handling* mode. If the fault handler

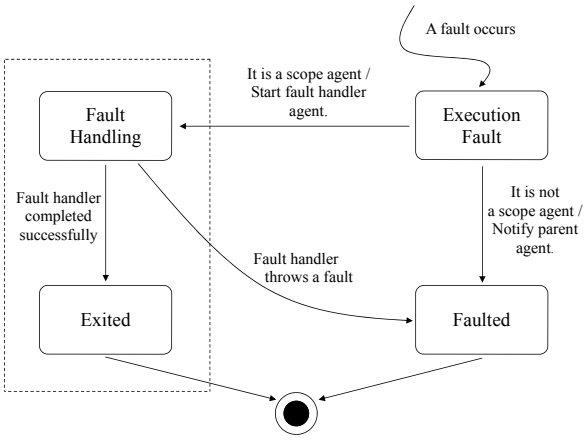



**Fig. 4.** Activity Execution Lifecycle: Fault Handling

finishes successfully, the scope agent enters the *Exited* mode indicating that this agent exited its execution with a successful fault handling process. The difference between a *scope* which has finished its execution in the *Completed* mode and a *scope* that has finished in the *Exited* mode is reflected by the way scopes are compensated, which we do not further address in this paper.

### 4.2 Pick activity: extended

The structured activities of the *core* (activity agents) are also refined to capture the fault handling behavior of BPEL. The well-defined activity execution lifecycle of $\mathsf{BPEL}_{\mathcal{AM}}$ (Figures 3 and 4) along with the fact that the fault handling behavior of BPEL is mostly centered in the *scope* activity, enable us to generally extend the behavior of structured activities by defining two new rules: HandleExceptionsInRunningMode and WaitForTermination. As an example, the pick agent program of Section 3.3 is refined as follows:

Pick Activity Extended

**PickProgram** $\equiv$
  $\mathsf{PickProgram}_{core}$
  **case** *execMode*(*self*) **of**
    *Running* $\rightarrow$ HandleExceptionsInRunningMode
    *ExecutionFault* $\rightarrow$ WaitForTermination
    *Faulted* $\rightarrow$ **stop** *self*

Activity agents react to a fault by informing their parent agent of the fault and stay in the *Execution-Fault* mode until they receive a notification for termination. If the parent agent is not a scope agent, the parent agent reacts in the same way and the fault is passed upwards until it reaches a scope agent. The scope agent handles the fault as described in Section 4.1, and sends a termination notification to its child agent. Upon receiving the notification, a sub-process agent that is waiting for a termination notification in

turn passes it to its child agents (if any) and enters the *Faulted* mode, where it then terminates. If a sub-process agent receives a termination notification while in its normal execution mode, it first enters the *Execution-Fault* mode and then reacts as if it were waiting for the notification.

The normal execution of activity agents in the *Running* mode is extended by the following rule:

Structured Activity Extended

**HandleExceptionsInRunningMode** $\equiv$
  **if** *faultExtensionSignal*(*self*) **then**
    **onsignal** $s$ : AGENT_EXITED
      *execMode*(*self*) := *ActivityCompleted*
    **otherwise**
      **onsignal** $s$ : AGENT_FAULTED
        TransitionToExecutionFault($fault(s)$)
      **otherwise**
        **onsignal** $s$ : FORCED_TERMINATION
          *faultThrown*(*self*) := *fault*($s$)
          PassForcedTerminationToChildren(*fault*($s$))
          *execMode*(*self*) := $emExecutionFault$

In the *Execution-Fault* mode, if a termination notification is received, the pick agent terminates its enclosing activity and goes to the *Faulted* mode. Analogously to the *Completed* mode, sub-process agents terminate their execution in the *Faulted* mode. For the complete extended pick agent program see [12].

## 5 Related work

There are various research activities to formally define, analyze, and verify Web Services orchestration languages. A group at Humboldt University is working on formalizations of BPEL for analysis, graphics and semantics [13]. Specifically, they use Petri-nets and ASMs to formalize the semantics of BPEL. However, the pattern-based Petri-Net semantics of BPEL [14] does not capture fault handling, compensation handling, and timing aspects; overall, the feasibility of verifying more complex business processes is not clear and still subject to future work. The ASM semantic model in [15] closely follows what we had presented in [16] with minor technical differences in handling basic activities and variables.

Formal verification of Web Services is addressed in several papers. The SPIN model-checker is used for verification [17] by translating Web Services Flow Language (WSFL) descriptions into Promela. [18] uses a process algebra to derive a structural operational semantics of BPEL as a formal basis for verifying properties of the specification. In [19], BPEL processes are translated to Finite State Process (FSP) models and compiled into a Labeled Transition System (LTS) which is used as a basis for verification. [20] presents a model-theoric semantics (based on situation calculus) for the DAML-S language which facilitates simulation, composition, testing, and verifying compositions of Web Services.

## 6 Conclusions

We formally define a BPEL abstract machine in terms of a distributed real-time ASM providing a precise and well defined semantic foundation for establishing the key semantic concepts of BPEL. Transforming informal requirements into precise specifications facilitates reasoning about critical language attributes, exploration of different design choices and experimental validation. As a result of our formalization, we have discovered a number of weak points in the LRM [12].

The dynamic nature of standardization calls for flexibility and robustness of the formalization approach. To this end, we feel that the ASM formalism and abstraction principles offer a good compromise between practical relevance and mathematical elegance — already proven useful in other contexts [6]. Our model can serve as a starting point for formal verification (considering formal specification as a prerequisite for formal verification). Beyond inspection by analytical means, we also support experimental validation by making our abstract machine model executable using the executable ASM language *AsmL* [21].

## References


1. Andrews, T., et al.: Business process execution language for web services version 1.1 (2003) Last visited Feb. 2005, `http://www-106.ibm.com/developerworks/webservices/library/ws-bpel`.
2. Organization for the Advancement of Structured Information Standards (OASIS): WS BPEL issues list. (2004) `http://www.oasis-open.org`.
3. W3C: Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language. (2003) Last visited May 2004, `http://www.w3.org/TR/2003/WD-wsdl12-20030303`.
4. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer-Verlag (2003)
5. ITU-T Recommendation Z.100 Annex F (11/00): SDL Formal Semantics Definition. International Telecommunication Union. (2001)
6. Glässer, U., Gotzhein, R., Prinz, A.: The formal semantics of sdl-2000: status and perspectives. Comput. Networks **42** (2003) 343–358
7. Börger, E., Glässer, U., Müller, W.: Formal Definition of an Abstract VHDL'93 Simulator by EA-Machines. In Delgado Kloos, C., Breuer, P.T., eds.: Formal Semantics for VHDL. Kluwer Academic Publishers (1995) 107–139
8. Glässer, U., Gurevich, Y., Veanes, M.: An abstract communication architecture for modeling distributed systems. IEEE Trans. on Soft. Eng. **30** (2004) 458–472
9. Gurevich, Y.: Sequential Abstract State Machines Capture Sequential Algorithms. ACM Transactions on Computational Logic **1** (2000) 77–111
10. Farahbod, R., Glässer, U., Vajihollahi, M.: Specification and Validation of the Business Process Execution Language for Web Services. In: Proc. of the 11th Int'l Workshop on Abstract State Machines, Springer-Verlag (2004)
11. Farahbod, R.: Extending and refining an abstract operational semantics of the web services architecture for the business process execution language. Master's thesis, Simon Fraser University, Burnaby, Canada (2004)
12. Farahbod, R., Glässer, U., Vajihollahi, M.: Abstract Operational Semantics of the Business Process Execution Language for Web Services. Technical Report SFU-CMPT-TR-2005-04, Simon Fraser University (2005) Revised version of SFU-CMPT-TR-2004-03, April 2004.

13. Martens, A.: Analysis and re-engineering of web services. To appear in 6th International Conference on Enterprise Information Systems (ICEIS'04) (2004)
14. Schmidt, K., Stahl, C.: A petri net semantic for BPEL4WS - validation and application. In Kindler, E., ed.: Proceedings of 11th Workshop on Algorithms and Tools for Petri Nets. (2004)
15. Fahland, D.: Ein Ansatz einer formalen Semantik der Business Process Execution Language for Web Services mit Abstract State Machines. Technical report, Humboldt-Universität zu Berlin (2004)
16. Farahbod, R., Glässer, U., Vajihollahi, M.: Specification and Validation of the Business Process Execution Language for Web Services. Technical Report SFU-CMPT-TR-2003-06, Simon Fraser University (2003)
17. Nakajima, S.: Model-checking verification for reliable web service. In: OOPSLA 2002: Workshop on Object-Oriented Web Services. (2002)
18. Koshkina, M., van Breugel, F.: Verification of Business Processes for Web Services. Technical Report CS-2003-11, York University (2003)
19. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Compatibility verification for web service choreography. In: Proceedings of the IEEE International Conference on Web Services (ICWS'04), IEEE Computer Society (2004) 738–741
20. Narayanan, S., McIlraith, S.A.: Simulation, verification and automated composition of web services. In: Proceedings of the eleventh international conference on World Wide Web, ACM Press (2002) 77–88
21. Farahbod, R., Gervasi, V., Glässer, U.: CoreASM: An extensible ASM execution engine. In: Proc. of the 12th Int'l Workshop on Abstract State Machines. (2005)

# Posters

# XML Schema-driven Generation of Architecture Components

Ali El bekai[1], Nick Rossiter[1]

[1]School of Informatics, Northumbria University,
Newcastle upon Tyne, UK
Email: ali.elbekai@unn.ac.uk, nick.rossiter@unn.ac.uk

**Abstract.** It is possible to code by hand an XSL stylesheet that validates an XML document against some or all constraints of an XML schema. But the main goal of this paper introduces general techniques as a technology solution for different problems such as (a) generation of SQL schema from XMLSchema, (b) generating XSL stylesheet from XMLSchema, and (c) XQuery interpreter generating. Each of the techniques proposed in this paper works by XMLSchema-driven generation of architecture components with XSL stylesheet. As can be seen the input is XMLSchema and XSL stylesheet and the output is generic stylesheets. These stylesheets can be used as interpreter for generating other types of data such as SQL queries from XQueries, SQL data, SQL schema and HTML format. Using XSL stylesheets we present algorithms showing how we can generate these components automatically.

## 1 Introduction

XML is fast emerging as the dominant standard for *representing* and *exchanging* information over the Internet [4,2]. If data is stored and represented as XML documents, then it should be possible to query the contents of these documents in order to extract, synthesize and analyze their contents. Also, it is possible to transform theses data to another format and to generate a component from the XML data. Originally, XML was created to meet the challenges of *data exchange* in Web applications or between applications and users, not for data presentation purposes. To deal with presentation issues, XML needs to be used in conjunction with stylesheets to be easily viewed on the web. For this reason, eXtensible Stylesheet Language was created. XSL (Extensible Stylesheet Language) is being developed as part of the W3C stylesheets activity [13,14]. It has evolved from the CSS language to provide even richer stylistic control, and to ensure consistency of implementations. It also has document manipulation capabilities beyond styling. Of course, designing “traditional” software transformation tools for that purpose can achieve such a task. However, the power of having a cross-platform and XML independent language would be lost. Precisely, isolating content from formatting needs to be considered, especially when dealing with Web based documents. Therefore, any method of transforming XML documents into different formats such as XML, HTML, SQL, flat files or WML needs to be tailored so that it can be used with different platforms/languages.

This paper introduces the technological solutions for different problems such as (a) SQL schema generation, (b) XSL stylesheet generation, and (c) XQuery interpreter generation automatically by transforming an XMLSchema through an XSLT.2.

## 2 Related Work

Bourret [2] noted that XML and its surrounding technologies have many facilities in common with real databases such as storage (XML documents), schemas (DTDs, XML schema languages), query languages (XQuery, XPath, XQL, XML-QL, QUILT, etc.), and programming interfaces (SAX, DOM, JDOM). On the other hand, XML lacks efficient storage, indexes, security, transactions and data integrity, multi-user access, triggers, queries across multiple documents, and so on. Aboulnaga *et al* [1] started a discussion in the XML community about characterizing and generating XML data. Provost [10] considered the most common patterns for document content constraints, and finds that XML Schema validation is only the first of several necessary layers. Bourret [3] summarized two different mappings. The first part of the process, generally known as *XML data binding*, maps the W3C's XML Schemas to object schemas. The second known as *object-relational mapping*, maps object schemas to relational database schemas. In [6,11] techniques are presented for querying XML documents using a relational database system, which enables the same query processor to be used with most relational schema generation techniques. Norton [8] presents an XSL as validation to validate XML document. W3C and Peterson describe a query processor that works for different schema generation techniques [12, 9]. Their work is done in the context of data integration, and the tables generated by each relational schema generation technique are specified as materialized views over a virtual global schema. In [5] a translation is presented of XQuery expression drawn from a comprehensive subset of XQuery to a single equivalent SQL query using a novel dynamic interval encoding of a collection of XML documents. In [7] an algorithm is presented that translates simple path expression queries to SQL in the presences of recursion in the schema in the context of schema-based XML storage shredding of XML relations.

As a result none of the approaches described above introduce an algorithm to generate a generic XSL stylesheet for transforming XML to SQL statements or an XSL stylesheet for transforming XQueries to SQL queries or for generating SQL schema by using XSL. We will next introduce general algorithms to generate these components automatically.

## 3 SQL Schema Generations

Basically, the DOM [12,16] is a specification that comprises a set of interfaces that allow XML documents to be parsed and manipulated in memory. The main interfaces for an application with DOM are: Node: the base type, representing a node in the DOM tree; Document: representing the entire XML document as a tree of Nodes (the DOM parser will return Document as a result of parsing the XML); Element: to rep-

resent the elements of the XML document; Attribute, representing an attribute of some XML element; Interface enabling setting/getting the value of that attribute; and Text: representing the text content of an element (i.e. the text between tags that is not part of any child element). The DOM tree is composed of nodes, each of which represents a parsed document. Based on these interfaces, we will use the XMLSchema parse file (DOM) as input in our algorithm to generate the components automatically. Now we will introduce an algorithm that can be used automatically to generate the SQL schema as output of an arbitrary XMLSchema. In particular, we present a translation algorithm that takes as input an XMLSchema and XSL stylesheet and produces a SQL schema as the output.

**Algorithm** generate SQL Schema **(XMLSchema** (DOM)**, XSL stylesheet)**
// Input XMLSchema, XSL stylesheet // Output SQL Schema (SQL DDL)
**1.** start
**2.** if the input arguments of the algorithm (XMLSchema, XSL stylesheet) exist
**2.1** Building DOM and parsing it
**2.2** if parsing XMLSchema is done then DOM will build dynamically
**3.2.1** perform XSL stylesheet transformation
**3.2.2** if transformation is done
**3.2.2.1** transformation processing (XMLSchema, XSLT)
**3.2.2.2** the XSL template adds SQL clause CREATE TABLE and matches the complexType name node of DOM
**3.2.2.3** for each complexType name create a separate SQL table
**3.2.2.3.1** create a primary key to each SQL table as table name followed by the string "_id"
**3.2.2.3.2** create a foreign key is also table name followed by the string "_fk" or identify child nodes and map them as foreign key to the SQL table as in our example
**3.2.2.3.3** If the root node (complexType) has parent/child nodes
**3.2.2.3.3.1** the XSL template matches and maps all child nodes as column names to created SQL tables
**3.2.2.3.3.2** the XSL stylesheet templates walk through DOM and matches all child nodes, attribute nodes data types and maps as column names to SQL
**3.2.2.3.3.3** report no parent/child node and terminate
**3.2.2.4** finishing the execution, SQL schema is generated (SQL DDL)
**3.2.3** report transformation errors and terminate
**3.3** report parsing errors and terminate
**4.** report reading errors and terminate
**5.** end/terminate

**Fig.1.** An algorithm for generating SQL schema from XMLSchema.

## 4 XSL Generations

Basically, it is possible to code by hand an XSL stylesheet that validates an XML document against some or all constraints of an XML schema. But in this section we introduce an algorithm as shown in Fig. 3 for generating an XSL stylesheet from an XMLSchema parse file (DOM) and as mentioned before the DOM tree is composed of nodes, each of which represents a parsed document. In other words this algorithm is the technology solution to the problem of generating an XSL automatically by transforming an XMLSchema through an XSLT. The result is a generic XSL stylesheet providing the mechanism to transform and manipulate XML data. Also the generated XSL stylesheet can be used to transform an XML document into another format such as XML to SQL statements and XML to HTML document.

## 5 The XQuery Interpreter Generation

Basically, the XQuery [15] is a language containing one or more query expressions. XQuery supports conditional expressions, element constructors, FOR, LET, WHERE,

RETURN (FLWR) expressions, expressions involving operators and function calls and quantifiers, type checking and path expressions. Some XQuery expressions evaluate to simple node values such as elements and attributes or atomic values such as strings and numbers. The syntax for retrieving, filtering, and transforming records uses FOR, LET, WHERE and RETURN clauses. A FLWR expression creates some bindings, applies a predicate and produces a result set. XQuery does not conform to the same conventions as SQL. XQuery and SQL share some similar concepts. Both languages provide keywords for projection and transformation operations (SQL SELECT or XQuery RETURN). SQL supports joins between tables and XQuery supports joins between multiple documents.

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/ 1999/XSL/Transform">
    <xsl:template match="complexType">
        DROP TABLE IF EXISTS <xsl:value-of select="@name" />;
        CREATE TABLE <xsl:value-of select="@name" />
        (<xsl:value-of select="@name" />_id INT NOT NULL,
        <xsl:apply-templates select="element" mode="@name" />
        PRIMARY KEY (<xsl:value-of select="@name" />_id) )
    </xsl:template>
    <xsl:template match="element" mode="@name">
        <xsl:value-of select="@name" />
        <xsl:apply-templates select="@type" mode="schematype" />
    </xsl:template>
    <xsl:template match="@type" mode="schematype">
        <xsl:variable name="type" select="." />
    <xsl:choose>
            <xsl:when test="$type='String'">VARCHAR2 (*)</xsl:when>
            <xsl:when test="$type='Date'">DATE NOT NULL</xsl:when>
            <xsl:when test="$type='Text'">TEXT</xsl:when>
            <xsl:when test="$type='Integer'">INTEGER (10,0) NOT NULL</
                xsl:when>
            <xsl:when test="$type='Image'">IMAGE</xsl:when>
            <xsl:when test="$type='Url'">URL</xsl:when>
            <xsl:when test="$type='Char'">CHAR (1)</xsl:when>
            <xsl:when test="$type='Sex'">SEX</xsl:when>
            <xsl:when test="$type='BLOB'">BLOB</xsl:when>
            <xsl:when test="$type='Country'">COUNTRY</xsl:when>
            <xsl:when test="$type='ReferenceType'">ReferenceType</xsl:when>
    </xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

**Fig. 2**. The generated XSL stylesheet for generating SQL schema from XMLSchema

Here are two simple examples: one with XPath type of a query and the other with FLWR expression. The single forward slash (/) signifies the parent-child relationship between elements. In tracing a path through a tree the expression starts at the root node and follows parent node and so on.

```
1) X / <collection>/<object>/<objectInfor>
2) For obj in <collection> do
      Where obj = <object>
      Return <object>
```

Finally in this section we introduce an algorithm for generating an XSL stylesheet from the XMLSchema to interpret the XQuery. In other words, this is the technology solution to the problem of generating automatically the XQuery interpreter by transforming an XMLSchema through an XSLT.

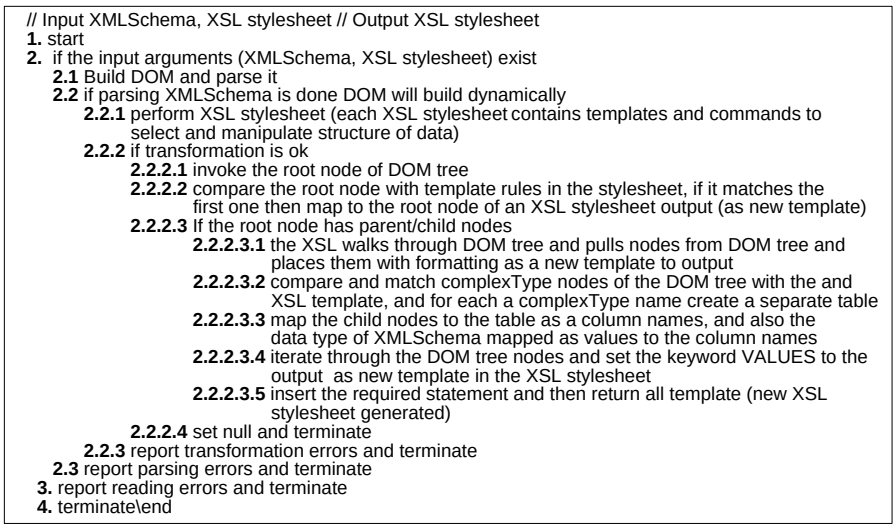



**Fig. 3.** An algorithm for generating XSL from XMLSchema.

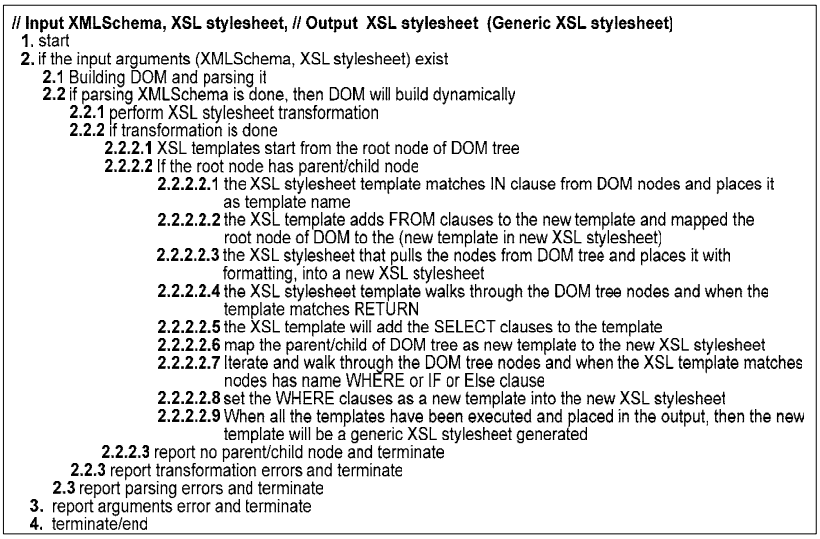



**Fig. 4.** Algorithm generating XSL from XMLSchema to transform XQueries to SQL queries.

## 6 Conclusion

The contribution of this work is that it introduces general techniques for generating SQL schema, XSL and XQuery using XMLSchema and XSL stylesheet, which (a) enables the use of these techniques for transforming XML data to SQL data and storing it in a relational database, (b) allows the user to present HTML format, and (c)

interprets XQueries (transforms XQueries to SQL queries) so that we can then retrieve and query data from the database. A potential cause for concern is that our general techniques may be less overlapping in implementation thus losing some efficiency.

However based on our prototype implementation in java, we have found that it is very quick to generate XSL stylesheets as an interpreter for different types of transformation such as SQL Schema and XQueries in to SQL queries. As we know it is possible to code by hand an XSL stylesheet that validates an XML document against some or all constraints of an XML schema and to generate an XSL stylesheet. However with our automated technique this task is easy, quick and less overlapping and we will use these generated components to integrate the (offline and online) components to satisfy our requirements. Also, we plan to extend these techniques to work with Multiple XMLSchema, not just single XMLSchema.

## References


1. Aboulnaga, A., Jeffrey F. Naughton, Chun Zhang. Generating Synthetic Complex-Structured XML Data. Fourth International Workshop WebDB'2001 (2001).
2. Bourret, R, XML and Database www.rpbourret.com/xml/XMLAndDatabases.htm (2004).
3. Bourret, R, Mapping W3C Schemas to Object Schemas to Relational Schemas., www.rpbourret.com/xml/SchemaMap.htm March (2001).
4. Chawathe S. Describing and Manipulating XML Data, Bulletin IEEE Technical (1999).
5. DeHaan D, D. Toman, M. Consens, and M. T. Özsu, A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding, Proc ACM Int Conf Management Data (SIGMOD'03), San Diego, 623-634 June (2003).
6. Florescu D, D. Koaamann. Storing and Querying XML Data using an RDBMS, IEEE Data Engineering Bulletin, **22** 27-34 (1999)
7. Krishnamurthy R., Venkatesan T.Chakaravarthy, Raghav Kaushik, Jeffrey F. Naughton, Recursive XML Schemas, Recursive XML Queries, and Relational Storage: XML-to-SQL Query Translation, ICDE (2004)
8. Norton Francis. Generating XSL for Schema validation http://www.redrice.com/ci/generatingXslValid, May 20, (1999)
9. Peterson David, Paul V. Biron, and Ashok Malhotra XML Schema 1.1 Part 2: Datatypes. W3C, Working Draft WD-xmlschema11-2-20040716, July (2004)
10. Provost W, XML Validation Architecture using XML Schema, XPath, and XSLT. (2002)
11. Shanmugasundaram J., E. Shekita, J. Kiernan, R. Krishnamurthy, E. Viglas, J. Naughton and Igor Tatarinov. A General Technique for Querying XML Documents using a Relational Database System. ACM SIGMOD Record, 30(3), (2001)
12. W3C, Document Object Model (DOM) Level 2 HTML Specification Version 1.0 W3C Recommendation 09 January. http://www.w3.org/TR/2003/REC-DOM-Level (2003).
13. W3C. Extensible Stylesheet Language (XSL) Version 1.0, W3C Candidate Recommendation. http://www.w3.org/TR/2001/REC-xsl-20011015/ October 15, (2001)
14. W3C XSL Working Group, W3C Recommendation on XSL Transformations (XSLT) http://www.w3.org/TR/xslt. (1999)
15. W3C.XQuery 1.0: An XML Query Language, W3C Working Draft July 23, (2004).
16. W3C.DOM Working Group, Document Object Model, http://www.w3.org/DOM/. (2004).

# Steering Model-Driven Development of Enterprise Information System through Responsibilities

Ming-Jen Huang and Takuya Katayama

School of Information Science, Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi-shi, Ishikawa, Japan
{m-huang, katayama}@jaist.ac.jp
http://kt-www.jaist.ac.jp/index-e.html

**Abstract.** OMG proposes the MDA that promotes the ideas of modeling in UML and transforming UML models to code. But UML is not universal for every domain and the direct translation approach of the MDA is not adequate. In this paper, we introduce REST, an idea of using responsibilities as contextual information to instruct machines to generate software systems. First, we give an overview of RESTDA - a software development architecture for business based on the concept of REST. Then we describe a domain-specific language - Business Models. It helps developers to describe a business from a document-processing perspective. We also introduce a rule-based validation of consistency within Business Models. Finally, we describe the transformation mechanism of RESTDA. Our approach provides machines higher intelligence to generate source code for different contexts.

## 1 Introduction

Model Driven Architecture (MDA) proposed by OMG [1] is a software development approach that promotes defining platform-independent models in UML and having machines to transform them into technology-specific code [2]. Its concept is based on two assumptions. First, UML is precise and expressive enough to describe problems we are interested in. It is also universal enough to describe any domain of problems. Second, all problems defined by every kind of UML modeling constructs should imply identical contextual information. For the MDA, UML becomes the master key to open every door to any solution.

With regard to the first assumption, different domains have different requirements. Thus, a domain-specific language (DSL) that is customized for a specific domain is more realistic and more productive [3]. With regard to the second assumption, considering the following example: does the case of implementing UML models of a car having four wheels equal to the case of a teacher having four students? By UML, they may be drawn identically in class diagrams or even sequential diagrams. For an effective model transformation mechanism, we do not only have to give machines syntactic and semantic information, but also the capability of reacting according to different contexts. To that end, we propose a conceptual idea, *Responsibility-Steering Model Transformation* (REST) to augment existing model-driven approaches.

REST is a conceptual idea of model transformation that is inspired by Responsibility-Driven Design, which is proposed by Wirfs-Brock [4]. She promoted the idea of designing a software system from responsibilities and devising role objects to collaboratively work together to assume these responsibilities [5]. In REST, we consider responsibilities of an abstraction level are *realized* by responsibilities of a level beneath. And realization of all responsibilities of all levels, combining with domain-specific languages, instructs machines to generate detailed implementation of different technology-specific code. The advantages of our approach are: (1) Responsibilities provide extra contextual information of domains under consideration. The problems like the example of car and teacher can be avoided. (2) By defining model transformation in terms of responsibilities of different levels, any change of requirements can lead to easy and reliable modification to the target system. (3) By formalizing responsibilities, the correct transformation can be ensured.

The purpose of our work is to devise a development architecture and to apply the idea of REST to the development architecture to solve the problems of the MDA mentioned above. In this paper, we introduce the development architecture for business called *Responsibility-Steering Development Architecture* (RESTDA).

The remainder of this paper is organized as follows. Section 2 gives the overview of RESTDA, the details of the DSL - Business Models, and the description of the rule-based consistent validation of BM. Section 3 describes the details of REST and its implementation in RESTDA. Section 4 gives the conclusions and future works.

## 2 Responsibility-Steering Development Architecture

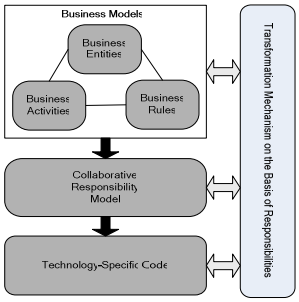



**Fig. 1.** Architecture of RESTDA

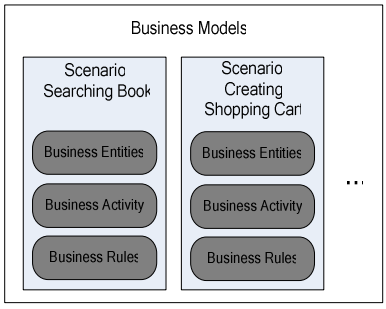



**Fig. 2.** Structure of Business Models

RESTDA is composed of a DSL to model concepts of business world and a model transformation mechanism between models and code. Fig. 1 shows the architecture of RESTDA. The DSL, *Business Models*, describes different business scenarios from the structural, behavioral, and constraint aspect. Definition of BM of a target system is transformed into a technology-neutral object model - *Collaborative Responsibility Model* (CRM) by machines with a business scenario as a unit. CRM does not contain details of technology-specific implementation but generalized software objects and responsibilities of these objects. By means of CRM, a system can be divided into many vertical-sliced parts, and each part can be transformed into different

technology-specific code that is most suitable to a situation. Instead of direct translation of meta-model to code, RESTDA applies the idea of REST, using syntactic, semantic, and contextual information, to instruct machines to transform BM to CRM and CRM to source code.

### 2.1 Business Models Description

BM defines the running of a business from three different views. *Business entities* describe the structural view. *Business activities* describe the behavioral view. *Business rules* describe constraints of business entities and business activities. Definition of BM of a target system has one or more scenarios which describe a possible situation of document processing (see Fig. 2).

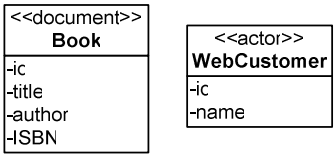



**Fig. 3.** A Sample of Business Entity Diagram

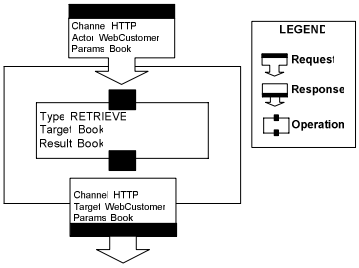



**Fig. 4.** A Sample of Business Activity Diagram

Business entities are roles that participate in a scenario. They are described in business entity diagram. A sample is shown in Fig. 3. Here, we borrow the drawing conventions from UML. There are two types of business entity, actor and document. Actor type entity represents human role in a scenario. In the diagram, it is displayed in stereotype `<<actor>>`. Document type entity is what is usually printed out as a formal or legal document in a business. In the diagram, it is displayed in stereotype `<<document>>`. Between business entities, they may have relations.

A business activity is a sequence of operations on which business documents are processed. A business activity has three parts, *request*, *operations*, and *response*. Request describes how the request is sent (Channel), who makes the request (Actor), and what information is carried by the request (Params). A single operation is an action operating on a document. It describes what type the operation is (Type), what document to operate on (Target), and what information to provide after completion of an operation (Result). There are four types of operation: CREATE, RETRIEVE, UPDATE, and DELETE. Operations can be linked sequentially to represent sequential operations. A business activity is described in a business activity diagram. As the exemplar Fig. 4 shown, the request is sent via `HTTP` and made by `WebCustomer`. `WebCustomer` should provide information of `Book` in the request. The business activity has a single operation to `RETRIEVE` information of `Book` and return resulting `Book`. The response is sent via `HTTP` to the `WebCustomer`. `WebCustomer` and `Book` are referred to the business entities of the scenario.

Business rules are constraints of business entities and business activities. For a business entity, business rules define the possible range of values of its properties. For a business activity, they define conditions of allowable activity requests or conditions of allowable operations, among other things.

### 2.2 Formalization and Implementation of Verification

The semantics of BM is formalized as predicates and implemented in a rule-based engine to verify validity of BM. These predicates are called *verification rules*. They are defined in terms of three basic constructs, *be* of business entity, *attr* of entity attribute, and *ba* of business activity. The types of business entity and activity are $DocumentType(be)$, $ActorType(be)$, and $BusinessActivityType(ba)$. Each construct has an identifier $ID(be)$, $ID(attr)$, and $ID(ba)$. Relations (*own* and *detailedBy*) between business entities are $Own(be_1,be_2)$ and $DetailedBy(be_1,be_2)$. Channel, actor, params of request are $RequestChannel(ba)$, $RequestActor(ba)$, and $RequestParams(ba)$. Type, target, and result of operation are $OperationType(n,ba)$, $OperationTarget(n,ba)$, and $OperationResult(n,ba)$ respectively ( $n$ denotes the sequence of operations). For example, $OperationType(1,ba)$ denotes the type of the first operation. Channel, actor, params of response are $ResponseChannel(ba)$, $ResponseTarget(ba)$, and $ResponseParams(ba)$ respectively.

There are five types of verification rules within BM. In this paper, we explain only the first type of verification rules - structural relation. In BM, business entities have two types of relation, own and detailedBy. For example, an actor type `Manager` owns a document type `MonthlySalesReport` and `MonthlySalesReport` is detailed by a document type `WeeklySalesReport`. Types of entity at two ends of a relation should be correct and they are represented as two rules:

1. Only an actor type entity can own a document type entity

$$\forall be_1,be_2\ Own(be_1,be_2) \Rightarrow ActorType(be_1) \wedge DocumentType(be_2)$$

2. Only a document type entity can be detailed by a document type entity:

$$\forall be_1,be_2\ DetailedBy(be_1,be_2) \Rightarrow DocumentType(be_1) \wedge DocumentType(be_2)$$

The verification rules are implemented in a rule-based engine, Jess [6]. Jess contains facts and rules. The collection of facts is information Jess knows. The collection of rules in Jess is a kind of actions that triggers under certain conditions [7]. Rules in Jess can be stated as "`if` $P$ `then` $A$". $P$ denotes a set of conditional facts. $A$ denotes a set of actions. $P$ is tested against all known facts. For example, if we know (1) a verification rule states that only an actor type entity can own a document type entity and (2) a fact states that `SalesStaff` (actor type) owns `PurchasingStaff` (actor type). If a Jess rule states "if (1) is not satisfied against all known facts, then displays a warning message." Since `PurchasingStaff` of (2) is not a document type, Jess would display a warning message.

## 3 Responsibility-Steering Model Transformation

REST is a conceptual idea of model transformation that uses responsibilities of different levels as contextual information to instruct machines to transform platform-independent models into technology-specific code. Real-world responsibilities of structural and behavioral constructs and constraints of a DSL are *realized* by generalized object responsibilities and the generalized object responsibilities are realized by responsibilities of technology-specific code, such as classes or components. In RESTDA, the realization of generalized object responsibilities is pre-defined. Developers only have to define (1) the responsibilities of BM and (2) how generalized object responsibilities realize these responsibilities for each scenario.

First, developers have to define real-world responsibilities from BM. A responsibility of any level always has a *holder* and a *receiver*. A holder represents a structural role which assumes the responsibility. A receiver represents a structural role that is affected by the consequence of the responsibility. Responsibilities of the same level are connected by holders and receivers. We use *Collaborative Responsibility Diagram* (CRD) to draw responsibilities, holders, and receiver as shown in Fig. 5. A collaborative responsibility diagram shows the structural and behavioral aspect of responsibility realization. To read the diagram, a rounded rectangle represents a responsibility and a rectangle represents a role. The left-hand role of a responsibility represents a holder and the right-hand role represents a receiver. A receiver of a responsibility could be a holder of another responsibility. The responsibilities are fulfilled from left to right one by one.

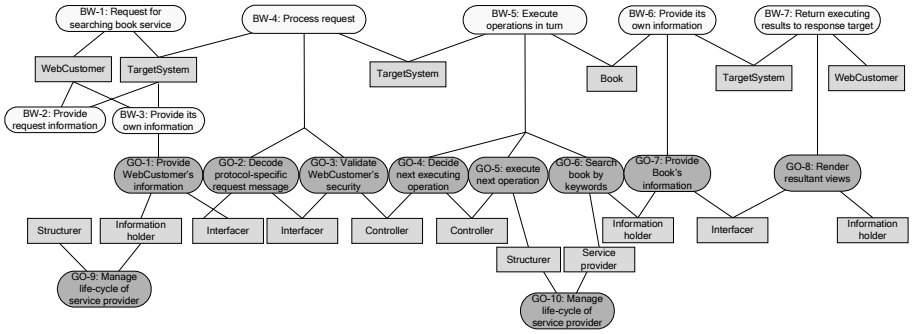



**Fig. 5.** A Sample of Collaborative Responsibility Diagram

Second, developers have to define how generalized object responsibilities realize the real-world responsibilities. It is a process of refinement by decomposing a real-world responsibility into smaller chunks. For example, the real-world responsibility "Process request" is realized by two generalized object responsibilities: "Decode protocol-specific message" and "Validate WebCustomer's security". Again, a holder and a receiver are assigned to a generalized object responsibility. They come from generalized objects. We borrow the concepts of role stereotypes from Responsibility-Driven Design. It defines six types of role: information holder, structurer, service provider, coordinator, controller, and interfacer [5]. A generalized software object represents a stereotype that assumes a set of generalized responsibilities. Developers have to contemplate types of responsibility and types of generalized object

simultaneously for each scenario. Responsibilities of generalized objects and their holders and receivers form CRM that are further transformed into Java code by Jess.

RESTDA predefines how a generalized object of CRM is transformed into one or more Java classes. The generation rules are also implemented in Jess in a code-template-generation fashion where the data for placeholders of code templates come from definition of CRM. These rules also define how different source code to generate for different responsibility definitions.

## 4 Conclusion and Future Work

In this paper, we introduced the software development architecture for business – RESTDA which is based on the idea of REST. The significance of the research is that domain experts can use BM to describe the running of a business without concerning any technology details. Instead of direct translation approach, the combination of syntactic, semantic, and contextual information of each level offers machines higher intelligence to generate software systems from platform-independent models.

With regard to future work, one is to formalize the concept of responsibilities Another is to use much expressive higher-order logic to quantify over predicates and to apply automatic theorem provers, such as HOL, to verify consistency of BM and responsibility realization [8,9].

## Acknowledgments

This research is conducted as a program for the "21st Century COE Program" by Ministry of Education, Culture, Sports, Science and Technology.

## References

1. MDA Guide Version 1.0.1. OMG. http://www.omg.org/docs/omg/03-06-01.pdf (2003)
2. Frankle, D.S.: Model Driven Architecture : Applying MDA to Enterprise Computing. Wiley, New York (2003)
3. Thomas, D.: MDA: Revenge of the Modelers or UML Utopia? IEEE Software, Vol. 21, No. 3, pp. 15 – 17 (2004)
4. Wirfs-Brock, R.: Object-Oriented Design: a Responsibility-Driven Approach. OOPSLA '89 Conference Proceedings, pp. 71 – 75 (1989)
5. Wirfs-Brock, R., McKean, A.: Object Design: Roles, Responsibilities, and Collaborations. Addison-Wesley, Boston (2003)
6. Jess v7.0a4. http://herzberg.ca.sandia.gov/jess/
7. Friedman-Hill, E.: Jess in Action. Manning: rule-based systems in Java. Manning, Greenwich, CT (2003)
8. Aoki, T., Katayama, T.: Unification and Consistency Verification of Object-Oriented Analysis Models. Asia-Pacific Software Engineering Conference, (1998)
9. Yatake, K., Aoki, T., Katayama, T.: Collaboration-Based Cerification of Object-Oriented Models in HOL. Verification and Validation of Enterprise Information Systems (2004)

# A Model-based Approach to Managing Enterprise Information Systems

Robert France[1], Roger Burkhart[2], Charmaine DeLisser[3]

[1] Colorado State University, Fort Collins, Colorado
france@cs.colostate.edu
[2]Information Systems, Deere & Company, Moline,Illinois, USA
BurkhartRogerM@johndeere.com
[3] University of Technology, Kingston, Jamaica
cdelisser@utech.edu.jm

**Abstract**. Organizations must evolve their information systems (IS) in order to adapt to changes in their environment or to maintain or enhance competitiveness. The use of modern application integration technologies (e.g., middleware) and advanced network technologies has resulted in IS that provide services at unprecedented levels, but at the price of becoming more complex and thus more difficult to evolve. By way of concrete examples, this paper focuses on the use of system models expressed in the Unified Modeling Language (UML) to effectively manage information systems assets. The system models capture critical information about an organization and are part of an overall framework called the Applic*ation Mapping Framework* or AMF. The AMF can be used by IT architects and planners to track applications, relate descriptions of system artifacts across different levels of abstraction and support redundancy, gap and impact analyses. The paper also identifies management roles needed to ensure that the AMF repository contains comprehensive and up-to-date models.

## 1 Introduction

The mission-critical role that Information Systems (IS) play in accomplishing business goals requires that they be managed and tracked as organizational assets. In this paper we describe a f*ramework* called the Application Mapping Framework (AMF), for organizing information about planned and deployed applications in an organization. The AMF is intended to support disciplined management and evolution of IS resources and enables business managers, information technology (IT) planners and architects, and application developers to (1) make decisions that minimize development risks and costs, (2) identify opportunities for cutting costs, and (3) identify new business opportunities. The AMF is more than just a static application portfolio. It provides services that can be used by IT planners and architects to support redundancy, gap and impact analyses. Proper use of the AMF will enhance the ability of an organization to maintain a corporate memory and utilize that memory to cost-effectively evolve its IS resources and business processes to meet business goals. The

AMF is intended to provide a single, accurate, organized source of information about business processes, applications, data and other IT resources.

An overview of the AMF architecture is presented in Section 2 of this paper. Types of analyses supported by the AMF and management functions required to build, use and maintain the AMF, are presented in Section 3. Section 4 explains by way of an example how the AMF could be used to support IS planning and evolution. The paper concludes with our views on the merits of using a model-based approach to IS management and an outline of our planned work in this area.

## 2 An Overview of the AMF

The AMF provides a logical architecture for a repository of information on applications and data within an organization. Its development is based on experience gathered on industrial projects that focused on developing application portfolios for organizations with a large and diverse set of distributed applications. The AMF specifies an application that is flexible in terms of the physical form or location of information could be captured and integrated in the framework. To help organize its wide range of topics and content, the information in the AMF is structured into a number of core views. Information in a core view can be further organized into sub-views.

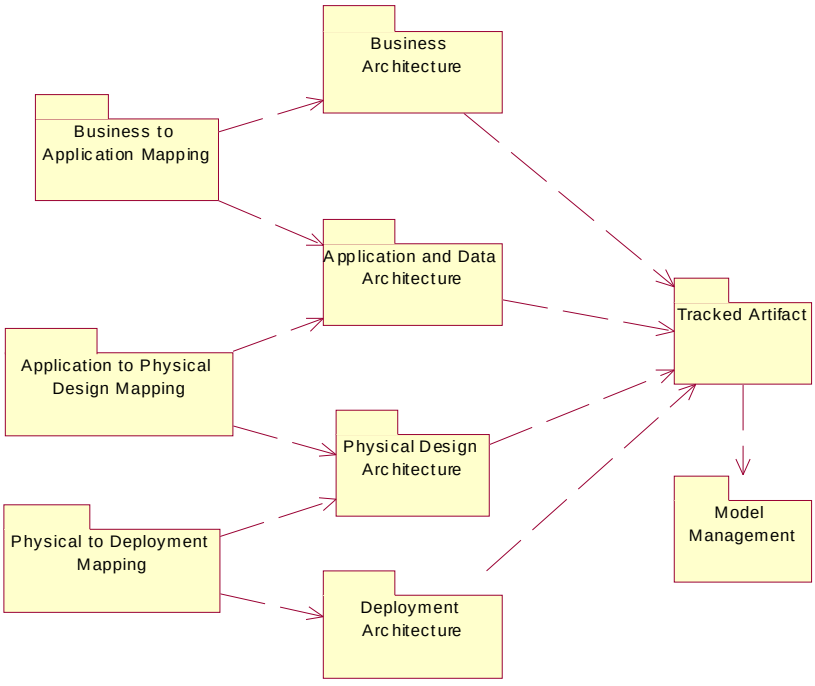



**Fig. 1.** Enterprise Application Map Structure

The UML (Unified Modeling Language) [1] is used to describe the structure of information in the AMF. An UML package is used to describe a view. UML Class Diagrams are used to describe the conceptual structure of information in a view, where a UML class represents a type of information item, a UML association represents a conceptual relationship between peer information item types, and an UML specialization relationship represents a further classification of an information item type. A view, represented as a package, contains a structure of packages (representing sub-views) and types (representing information item types).

At the top level, the AMF is organized into three views.

- The *IT Planning View* contains information pertaining to ongoing and planned IS projects, and includes information on tactical and strategic plans. The IT Planning View is intended to support the work of IT planners and project managers.
- The *Asset View* contains information about enterprise-wide and domain-specific reusable business artifacts. This view is intended to support systematic reuse of development experiences across an organization.
- The *Enterprise Application Map* is the central component of the AMF and contains information about the current and planned information system resources (e.g., applications, data, processes, roles) that are tracked within an organization. The previously mentioned views utilize information within this view (as indicated by the dependency relationships – the dashed arrows – between the packages).

This paper focuses on the Enterprise Application Map. The information in the Enterprise Application Map is organized into the following primary views (see Figure 1):

- *Business Architecture*: This view contains information about the business processes and entities that are tracked by an organization.
- *Application and Data Architecture*: This view contains information about the logical (i.e., technology-independent) aspects of applications and data. The information includes descriptions of the IS artifacts (applications and data) as they exist, as well as plans for evolving the artifacts. Descriptions include models and artifact metadata.
- *Physical Design Architecture*: This view contains information about the physical design of applications and data, that is, it presents a technology-specific view of applications. This view allows one to track the technologies that are used to implement applications and data.
- *Deployment Architecture*: This view contains information about the deployment and usage of applications within an organization. Information pertains to the "as is" deployment and usage of applications and data, as well as to planned deployments and usages.

Relationships between concepts across these views are described by the Mapping Packages:

- *Business to Application Mapping*: This package links elements in the Business Architecture view to the Application and Data Architecture view. The mappings provide traceability of business concepts to logical (platform independent) application concepts.

- *Application to Physical Design Mapping*: The mappings in this package provide traceability of logical application elements to physical (platform-specific) design elements.
- *Physical to Deployment Mapping*: The mappings in this package provide traceability of physical design elements to the artifacts to their deployed forms.

The other packages of information in the Enterprise Application Map contain information that is orthogonal to the packages described above.

- *Tracked Artif*act: This package contains information about properties that are common to artifacts that are tracked in the AMF. Currently, this includes only information pertaining to versioning of artifacts.
- *Model Management*: This package contains information about models, groupings of model elements used to present views of applications, the tools used to display models and the organizational roles responsible for maintaining the views.

## 3 Using the AMF

This section outlines the kinds of analyses that are supported and the management roles that are recommended for effective management and evolution of the AMF.

### 3.1 Management Roles

Effective use of the AMF by business analysts, architects and system developers is possible only when the contents of the AMF are relevant, properly packaged, easily retrieved, current and accurate. The following are recommended management roles that address issues related to the relevancy, accuracy, and usability of the AMF:

- *Content Manager*: Responsible for packaging, cataloging, and updating AMF contents.
- *Content Collector*: Responsible for collecting candidate contents.
- *Content Certifier/Evaluator*: Responsible for evaluating and certifying candidate AMF contents. The evaluation is carried out to determine, for example, the accuracy, relevance, and currency of candidate content.
- *Content Disseminator*: Responsible for promoting and facilitating the use of the AMF.
- *AMF Strategic Planner*: Responsible for developing and maintaining plans for evolving the AMF. This involves analyzing the usage of the AMF, analyzing repository contents (e.g., identifying content with diminishing returns), and identifying opportunities.

### 3.2 User Roles

Users of the AMF can be classified in terms of the roles they play in system and business process management and development. Below we list the roles and the types of interactions they can have with the AMF.

- *Business Analyst*: Responsible for defining, documenting and updating business processes.
- *IS Architect*: Responsible for planning and managing the integration and evolution of IT systems that support business processes.
- *System Architect*: Responsible for designing and managing the evolution of a particular system.
- *System Developer*: Responsible for implementing system designs and changes.

### 3.3 Model-based Analyses

A sample set of IT project planning activities supported by the AMF are listed below:

- *Impact Analyses*: During project planning one needs to determine, among other concerns, how the system to be developed impacts other systems, what resources are required and available for the project, what parts of the system functionality can be provided by existing system components and what parts need to be built or acquired. The AMF can be used to support impact analysis, determining the impact of change on the organization's ability to effectively meet business needs. The relationships among the artifacts in the AMF (for example, data/object create, read, update, and delete relationships between applications and data/objects) can be used to determine the impact of planned changes and new features on existing applications and data and on other current and planned IT projects.
- *Gap Analyses*: The AMF can also be used to support gap analyses. As new processes and system functionality are developed, gaps in the existing integrated system need to be identified and filled. Gap analysis is concerned with determining the missing functional and process elements that need to be present in order to implement new functionality of processes. The repository can be used to determine what parts of a system are under development or already exists, and what parts need to be obtained from outside vendors or be built in-house.
- *Redundancy analyses*: As an organization's pool of systems grows, the need to identify redundancies to reduce inefficiencies and avoid conflicts arising from multiple representations of a single concept across an organization becomes evident. Redundancy analysis is concerned with identifying systems that provide similar services. The repository can be used to determine whether proposals for new system features can already be met by existing systems and to determine wasteful overlaps in system functionality.
- *Reuse analyses*: Order-of-magnitude improve-ment in productivity and system quality can be accomplished if developers reuse product experiences. A well-managed integrated system can form the basis for identifying potentially reusable experiences across an organization. Reusability analysis is concerned with identifying potentially reusable artifacts. Commonality analyses can be carried out on the repository to identify organization-wide and domain-specific patterns that can be packaged for reuse (e.g., as product frameworks, components, reusable models).

## 4 An order fulfillment process scenario

The following scenario, though fictional, provides a realistic view of how the AMF can be used to support IT planning and system evolution within an organization. The Commercial Equipment (CE) Division of a fictitious organization has acquired a new distribution channel that is located overseas. The need to adhere to reporting regulations and other standards (e.g., customer addressing) in force within the foreign territory requires CE to reengineer its order fulfillment processes and systems.

In this scenario the AMF contains a web of artifacts ranging from business models of the processes to documents describing the deployment and usage of applications, and the computing infrastructure that currently support the processes. The AMF is accessed through interfaces that provide reporting functions and browsing starting points that are particular to the roles of the individuals accessing the repository. For example, the Business View interface of the AMF provides business analysts with a business-oriented view of the repository contents from which they can drill down to more system-specific views if required.

In the absence of an AMF, analysts, planners and developers have the challenge of locating, relating, and analyzing possibly poorly documented information about the current order fulfillment processes and supporting systems within CE. They may even have to revert to source code analysis. These activities are expensive, error-prone, and time-consuming. More importantly, such an environment is not conducive to the development of systems that fully exploit resources that can significantly reduce the cost of development without sacrificing system quality.

### 4.1 Business Analysis

The business analyst is responsible for defining an order fulfillment business process that will handle the orders of the new distribution channel. To carry out this task the analyst needs to (1) consider the impact of the proposed process solutions on existing processes and systems, and (2) identify possible opportunities for exploiting current system resources in the execution of the new processes, in order to define a cost-effective and realizable process.

As a starting point, the analyst uses the AMF to determine the location of documents that describe the current CE order fulfillment processes. Using the documentation reporting facility of the Business View interface, the analyst locates information on order fulfillment processes. A partial view of the table that is displayed as a result of the interaction with the AMF is shown below:

**Table 1.** Relationships between Processes and the Responsible Organizational Roles

| Subject Area | Business Activity | Responsible Organization |
|---|---|---|
| Order | Order Fulfillment | Org1 |
| | Order Entry | Org2 |
| | Order Routing | Org2 |
| | Order Management | Org3 |

The order fulfillment processes are contained in the Order subject area and consists of business sub-activities. Clicking on an activity name in column 2 takes the analyst to a page that contains model(s) of the activity. These models can be expressed as Activity Diagrams, Interaction Diagrams, and/or Use Cases. Using these links the analyst can not only access models that help in understanding the processes, but also use the information to identify models that are impacted by the change and that can be reused to describe the changed process. The analyst also needs to work with the owners of the process descriptions that will be impacted by the change. Clicking on the items in the third column of Table 1 results in a page that displays contact information for business process owners.

Table 2 is a partial view of the table that is displayed when Order Entry in column 2 of Table 1 is selected (in this case the process models are organized by the types of orders processed):

**Table 2.** Process Model Table for Order Entry

| Business Activity | Order Entity Type | Essential Process Model | Process Realization Model |
|---|---|---|---|
| Order Entry | Domestic Dealer Order | Ess-.mdl | Real-OD.mdl |
| | Export Order Region 1 | Ess-E1.mdl | Real-E1.mdl |
| | Export Order Region 2 | Ess-E2.mdl | Real-E2.mdl |

There are a number of variants of the Order Entry process, each determined by the type of order it processes. Selecting an order type in column 2 of Table 2 results in a page that describes the order type. Columns 3 and 4 contain pointers to models of the processes. An essential process model describes a process in terms of externally observable effects (i.e., effects that are observable by users of the business processes – the external view), and a process realization model describes a process in terms of how the activities are carried out (the internal view).

The analyst also needs to determine the business entities that are impacted by the change. To support this task the AMF can be used to produce the following table:

**Table 3**. Trace relationships between business activities and business entities

| Business Activity | | Business Entity | Access Type | Responsible Organization |
|---|---|---|---|---|
| Order Fulfillment | | | | Org 1 |
| | Order Entry | Order<br>FDD<br>… | Create<br>Create<br>… | Org2<br>Org2 |
| | Order Routing | Supplier<br>… | Update<br>… | Supp |
| | Order Management | Customer Account<br>… | Update<br>… | Customer Dept |

Selecting items in column 1 of Table 3 results in a page that shows the realization process models indicating which entities are created, accessed, updated and deleted by the business activity. Column 2 lists the business entities that are manipulated by the business activities, and column 3 specifies the type of access (Create, Read, Update, Delete).

The analyst also needs to have an idea of the order fulfillment systems and databases that would be impacted by the change in order to identify a cost-effective process solution. Another table (not shown), can show the relationship between the business activities and the systems and databases that support the activities. Selecting on the items in System and Database columns can link the analyst to a page that contains descriptions of the artifacts, contact information for the owners of the artifacts, and pointers to more detailed information about the systems and databases.

## 5 Related Work

Other frameworks for information systems architecture are being used today, most notably being the Zachman Framework (ZF), the Four+one framework and the RM-ODP. Each has its own merit providing developers of new systems architectural options for conceptualizing and designing. Zachman Framework is pre-object and reflects a structured approach to development. It consists of a thirty-six-cell matrix covering the perspectives of different stakeholders and aspects of the architecture. It is seen as the best way to conceptualize all the elements of a system but has been criticized as being process-heavy, requiring years to create. Ambler [3] suggests ways in which ZF can be used in an agile manner.

The RM-ODP [2] is rooted in object analysis, and covers five viewpoints enterprise, information, computational engineering and technology. Evitt [4] points out that the viewpoints are abstract and do not reflect the concerns of specific stakeholders as the ZF.

The AMF being proposed provides a lower level of detail than the ZF and RM-ODP. Whereas the frameworks mentioned above can be used for developing *new* systems, the AMF is intended for use as a lightweight means to document *existing* systems and the way they relate to each other. It is to be used as a management tool for identifying gaps, redundancies and reuse opportunities, and to be able to perform impact analysis. The AMF can be used within the context of both the ZF and the RM-ODP.

## 6 Further work

The AMF can provide a comprehensive representation of an enterprise's business and information systems and the means to conduct relevant queries and analyses. The proposed business architecture, application and data architecture etc. serve to define a workable structure for organizing, managing, analyzing and evolving enterprise information systems. Populating this framework with suitable, well placed and accurate business and information system design and implementation models

however, requires the involvement of skilled modelers, the formulation of and adherence to standards of operating that will guarantee capture of accurate information in a timely manner. The discipline required to make the use of the AMF a success will ultimately result in improved practices, processes and tools and to a more mature use of IT.

Our next step is to validate the AMF by using it to develop an IS repository for an industrial partner. We are currently evaluating different development environments for hosting, populating and querying an AMF repository. We will then develop and deploy a prototype repository infrastructure and evaluated its usage. The experience we gain will help refine the architecture and give insights into the types of mechanisms needed to seamlessly integrate AMF related activities and IS development and planning activities.

## References

1. The Object Management Group (OMG): Unified Modeling Language: Superstructure, the OMG, {http://www.omg.org}, Version 2.0, Final Adopted Specification, August, 2003
2. Open Distributed Processing Reference Model: ISO/IEC IS 10746. http://www.joaquin.net/ODP. Visited 3/01/2005
3. Ambler, S.W., Architecture and Architecture Modeling Techniques (2002) http://www.agiledata.org/essays/enterpriseArchitectureTechniques.html Visited 12/11/2002
4. Evitts, P. : A UML Pattern Language. MacMillan Technical Publishing (2000)

## Author Index